Universidad Carlos III de Madrid
Escuela Politécnica Superior
Grado en Ingeniería Informática

Trabajo de Fin de Grado

Aceleración de un procesador de imagen de resonancia magnética mediante GPUs

Autor: Jesús Cristina Iglesias
Tutor: Francisco Javier García Blas

Leganés, Madrid, España
21, Junio 2017

Índice general

Índice de tablas

Índice de figuras

Agradecimientos

Primeramente, se agradecerá a aquellos profesionales que tuvieron una contribución activa y continua en el desarrollo de este trabajo. Entre ellos, cabe destacar a mi tutor Francisco Javier García Blas, por haberme guiado en la realización de este proyecto con una disponibilidad plena. También agradecer a los miembros del grupo de investigación ARCOS, por permitirme el acceso a su *clúster* de cómputo para el ámbito de pruebas, evaluación y rendimiento, y por sus consejos en los últimos retoques que tanto marcan la diferencia.

Además, cabe destacar una especial mención a aquellos compañeros que me aportaron información y sabiduría a la hora de llevar a cabo este trabajo, como José Manuel Fernández Ruiz, Carlos Contreras Sanz, Alejandro García-Cantarero Alañón y José Ignacio Coig-O'Donell Velasco.

Finalmente, incluir a Sara Colás Carpallo, estudiante de enfermería de la UAM, por rellenar mis lagunas de conocimiento sanitario.

Resumen

Este documento presenta el Trabajo de Fin de Grado titulado “Aceleración de un procesador de Imagen de Resonancia Magnética mediante GPUs”.

Primero se realizó un estudio del estado del arte sobre las tecnologías y métodos que pueden ser útiles para demostrar la hipótesis propuesta. En este estudio se comparan las principales bibliotecas de álgebra lineal y las tecnologías que permiten la ejecución de código sobre GPUs. En segundo lugar, se realizó una serie de reuniones con el tutor del proyecto para definir los necesidades. Estas necedades fueron plasmadas en la descripción de requisitos. Una vez que los requisitos fueron aceptados por el tutor, se realizó el diseño e implementación de la solución. Seguidamente, se diseñaron las pruebas necesarias para verificar que el sistema cumple con las especificaciones acordadas con el tutor.

Para demostrar la hipótesis definida por el título del presente proyecto, se realizó una evaluación de rendimiento, en la cual los resultados de la ejecución de la aplicación sobre dispositivos GPU demuestra las ventajas del prototipo implementado. Estos resultados demuestran que la hipótesis es real y que los tiempos de ejecución son altamente reducidos en comparación con la versión anterior de la aplicación.

Este documento también muestra una planificación del proyecto, el presupuesto asociado al mismo, un capítulo dedicado a los aspectos legales referentes al proyecto, y un capítulo dedicado al impacto socio-económico del proyecto.

Capítulo 1

Introducción

Este capítulo muestra una visión general sobre el proyecto basado en la aceleración de un Procesador de Imagen por Resonancia Magnética (RMI) utilizando GPUs. En este trabajo se explica la motivación que lleva a realizarlo y los objetivos que se desean alcanzar al finalizar. De igual manera, se desglosa la estructura que seguirá el documento y, finalmente, una lista de los términos y acrónimos más utilizados.

1.1. Motivación

A lo largo de los años, las mejoras en la tecnología y los avances en la ciencia han permitido al mundo de la medicina realizar diagnósticos de una forma más rápida, segura y menos invasiva para los usuarios del Sistema de Salud (en adelante SS). Gracias al uso de los imanes en la medicina moderna es posible generar imágenes del interior del cuerpo humano sin la necesidad de utilizar una técnica cruenta para el paciente.

Esta técnica, denominada IRM (Imagen por Resonancia Magnética) o MRI (del inglés, Magnetic Resonance Imaging) permite obtener imágenes de los tejidos blandos del cuerpo humano utilizando para ello un potente imán. El uso de métodos de reconstrucción intravoxel basados en datos de resonancia magnética por difusión, permite realizar un estudio sobre la organización de la sustancia blanca del cerebro. Un estudio posterior de las conexiones de la fibras del cerebro permite el diagnóstico de enfermedades tales como la esquizofrenia o trastornos bipolares.

Las imágenes tomadas mediante resonancia magnética pueden considerarse matrices de números. Un

computador convencional posee una unidad denominada CPU (del inglés, Central Processing Unit). Las CPUs realizan operaciones secuenciales sobre un dato a la vez. Sin embargo, existen otros tipos de dispositivos denominados GPU (del inglés, Graphics Processor Unit), los cuáles están especializadas en realizar la misma operación sobre varios datos a la vez. Esta técnica, conocida como SIMD (del inglés, Single Instruction Multiple Data), resulta ser beneficiosa para realizar cálculos sobre matrices. Como se ha indicado anteriormente, las imágenes tomadas por las resonancias pueden considerarse matrices de números.

En el caso particular de este proyecto, al utilizar las GPU para realizar los cálculos sobre las imágenes tomadas por resonancia magnética, resulta en una mejora en los tiempos de cálculo y por tanto del tiempo de ejecución.

Por otro lado, *Matlab* ha sido elegido en muchos campos de la ciencia como lenguaje estándar. Aunque permite un rápido despliegue de prototipos, *Matlab* debido a su naturaleza es un lenguaje que es fuertemente dependiente de su propio motor de ejecución, lo que hace que aunque se utilice sobre equipos de altas prestaciones, no llegue a ofrecer los resultados esperados.

Por tanto, la principal motivación de este trabajo es la de que gracias a un lenguaje de alto nivel como C++ y a técnicas de programación en GPU se puedan conseguir mejoras en los tiempos de ejecución de un procesador de imagen de resonancia magnética.

1.2. Objetivos

El objetivo principal del trabajo es **analizar la posibilidad de aceleración de un Procesador de Imagen por Resonancia Magnética mediante la utilización de GPUs**. Como objetivos secundarios dentro del anterior se pueden encontrar:

- Portar la aplicación existente a un lenguaje más optimizado y compatigle con GPUs.
- Usar bibliotecas de álgebra lineal para optimizar los cálculos matriciales de la aplicación.

Este documento presenta el diseño llevado a cabo para realizar la mejora sobre el código. Por supuesto, las imágenes generadas con la mejora ofrecida deben ser totalmente iguales a las generadas con el método anterior.

1.3. Estructura del documento

El presente documento tiene la siguiente estructura:

- **Introducción (Capítulo 1):** Primer capítulo del documento que se centra en presentar el proyecto, explicar los objetivos y la motivación del mismo. Incluye también una enumeración de la estructura del documento, acrónimos utilizados y las definiciones de los términos más utilizados.
- **Estado del arte (Capítulo 2):** El segundo capítulo del documento presenta un estudio del entorno que influye al sistema del proyecto. Dentro del mismo se realiza un análisis sobre si existen aplicaciones similares, y análisis sobre las tecnologías que utiliza.
- **Análisis del sistema (Capítulo 3):** En el tercer capítulo del documento se realiza un estudio sobre las funcionalidades que debe presentar el sistema desarrollado.
- **Diseño del sistema (Capítulo 4):** El objetivo del cuarto capítulo es el de expresar las clases y las funciones de esas clases que van a formar parte del proyecto.
- **Implementación (Capítulo 5):** Los aspectos más importantes sobre el desarrollo del sistema y la explicación a un nivel más bajo de las clases y sus funciones, son el contenido del quinto capítulo.
- **Plan de pruebas (Capítulo 6):** En el sexto apartado del documento se muestran el plan de pruebas y las pruebas realizadas para comprobar el funcionamiento correcto del sistema.
- **Evaluación y rendimiento (Capítulo 7):** El séptimo capítulo, expresa la evaluación hecha sobre la aplicación y el rendimiento final obtenido en comparación con la aplicación inicial.
- **Planificación y presupuesto (Capítulo 8):** La planificación y estudio del presupuesto asociado al proyecto son el tema del octavo capítulo.
- **Marco regulador e impacto socio-económico (Capítulo 9):** En el noveno capítulo del documento se explica el marco legal, y las cuestiones relacionadas con la propiedad intelectual que afectan al proyecto. La última sección de este capítulo referencia al impacto social del proyecto.
- **Conclusiones y trabajos futuros (Capítulo 10):** En el último capítulo del documento se expresan las conclusiones extraídas de la realización del proyecto y la dirección a seguir para los trabajos futuros relacionados.
- **Anexo 1:** Resumen de 14 páginas del proyecto, escrito en inglés.

1.4. Glosario

En este apartado se mencionarán y explicarán algunos términos técnicos que aparecen en el documento:

- **Biblioteca:** agrupación de funcionalidades, que mediante una interfaz pueden ser utilizadas en un código.
- **CPU:** unidad central de procesamiento. Es la parte más importante de un computador.
- **GPU:** unidad de procesamiento gráfico. Esta especializado en operaciones con matrices de datos.
- **Imagen por Resonancia magnética:** imagen que ha sido generada mediante la técnica de Resonancia Magnética.
- **Pragma:** en este documento el término *pragma* hace referencia a unas sentencias de *OpenMP* que permiten que la parte del código que engloban tal *pragma*, sea ejecutado en paralelo.
- **Referencia:** en este documento el término referencia si hacemos alusión a los parámetros de una función, estamos indicando que se pasa una variable que apunta a la misma dirección de memoria que la variable original, de modo que cualquier cambio sobre la variable recibida se verá reflejado en la variable original.
- **Slice:** en este documento se refiere al método de lectura de datos por el cuál se lee una parte del volumen de datos. Esta parte corresponde con un plano del volumen de datos, es decir, fijar una de las tres dimensiones y coger el plano resultante de las otras dos.
- **Speedup:** en este documento nos referimos al *speedup* a la mejora en tiempo de ejecución que se obtiene entre el código desarrollado ejecutándose sobre GPUs y el código anterior.
- **Template:** en este documento el término *template* hace referencia a una declaración de una función que puede admitir cualquier tipo de dato cuando es invocada.
- **Tile:** la traducción literal del inglés es baldosa, pero en este documento se usa en el Capitulo 7 para referirnos a la submatriz de la matriz de datos que utiliza la GPU para realizar los cálculos.
- **Volume:** en este documento se refiere al método de lectura de datos por el cuál se lee todo el volumen de datos a la vez.
- **Voxel:** unidad mínima de un objeto tridimensional, es decir, corresponde a un píxel en un objeto de tres dimensiones.

1.5. Acrónimos

La siguiente lista muestra la definición de los acrónimos utilizados en el documento.

- **BLAS:** del inglés, Basic Linear Algebra Subprogram.
- **CPU:** del inglés Central Processing Unit.
- **CUDA:** del inglés, Compute Unified Device Architecture.
- **DSI:** del inglés, Diffusion Spectrum Imaging.
- **DTI:** del inglés, Diffussion Tensor Imaging.
- **EISPACK:** del inglés, Eigen System PACKage.
- **FFT:** del inglés, Fast Fourier Transform.
- **GPU:** del inglés Graphics Processor Unit.
- **HARDI:** del inglés, High Angular Resolution Diffusion Imaging.
- **IFFT:** del inglés, Inverse Fast Fourier Transform.
- **Intel MKL:** Intel Math Kernel Library.
- **IRM:** Imagen por Resonancia Magnética.
- **LAPACK:** del inglés Linear Algebra PACKage.
- **NIfTI:** del inglés Neuroimaging Informatics Technology Initiative.
- **PSF:** del inglés Point Spread Function.
- **RMI:** del inglés Magnetic Resonace Imaging.
- **RUMBA-SD:** del inglés, Robust an Unbiased Model-BAsed Spherical Deconvolution.
- **SIMD:** del inglés, Single Instruction Multiple Data.
- **SS:** Sistema de Salud.
- **2D:** dos dimensiones.
- **3D:** tres dimensiones.
- **4D:** cuatro dimensiones.

Capítulo 2

Estado del arte

En este capítulo del documento se van a explicar varias tecnologías que permiten utilizar C++ como un lenguaje para realizar cálculos de álgebra lineal. Además, se explicarán varias tecnologías que permiten que se realicen cálculos de matrices sobre las GPUs en vez de sobre las CPUs.

2.1. Trabajos relacionados

El tutor del proyecto ya ha realizado trabajos de portabilidad de aplicaciones en busca de una mejora. En el artículo de investigación [3], el tutor presenta una serie de soluciones sobre una aplicación escrita en *Matlab*. En dicho artículo de investigación realizan una traducción del algoritmo RUMBA-SD, de un conjunto de herramientas de imagen médica llamada HARDI. Además, el tutor presenta pruebas de rendimiento con distintas soluciones y finalmente se comprueba que la versión que utiliza Arrayfire es la que mejor resultados ofrece debido al uso de GPUs.

2.2. Intravoxel Fiver Reconstruction (RUMBA)

Es un método de reconstrucción que permite determinar la organización estructural de la sustancia blanca del cerebro. Conocer las conexiones entre las fibras nerviosas del cerebro ayuda a diagnosticar enfermedades mentales como la esquizofrenia y trastornos bipolares.

$$D = \begin{bmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{bmatrix}$$

Figura 2-1: Matriz del tensor de difusión.

Tal y como se explica en el documento [13] y en el artículo de investigación [3], en los tejidos biológicos, la difusión de las moléculas de agua no se realiza de forma similar en todas las direcciones debido a la presencia de obstáculos o barreras. Este fenómeno puede ser explicado cuantitativamente utilizando el tensor de difusión DTI. La representación de este tensor es una matriz simétrica 3x3 positiva que puede observarse en la Figura 2-1.

Este tensor de difusión se puede visualizar como un elipsoide, en el cuál las orientaciones de los tres ejes principales se definen por su vectores propios y la longitud de los ejes viene definida por sus valores propios.

El modelo del tensor de difusión resulta adecuado en el caso de que cada *voxel* del cerebro presente un solo grupo de fibras paralelas (ver panel a de la Figura 2-2). En este caso, el eje principal del elipsoide correspondería con la orientación del grupo de fibras. Sin embargo, en regiones donde se cruzan varias fibras, el modelo del tensor de difusión no permite calcular la orientación de éstas fibras (ver panel b de la Figura 2-2).

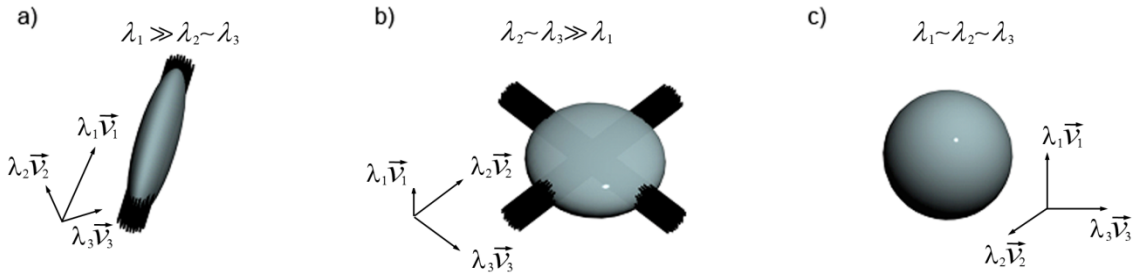

Figura 2-2: La organización microestructural puede ser revelada por el elipsoide de tensor de difusión.

Debido a las limitaciones que presenta DTI, se ha impulsado el desarrollo de protocolos de muestreo, modelos de difusión y técnicas de reconstrucción. Uno de estos métodos es RUMBA-SD cuyo objetivo es el de obtener para cada *voxel* una función de distribución de la orientación de las fibras cerebrales.

Como se muestra en la Figura 2-3, para facilitar la visualización, se asigna el código de color de la superficie de acuerdo con la dirección de difusión ($[x, y, z]-[r, b, g]$, donde r = rojo representa la probabilidad a lo largo de la orientación de izquierda a derecha, b = azul representa superior-inferior orientación, y g = verde representa la orientación antero posterior).

Figura 2-3: Función de distribución de orientación.

2.3. Lenguaje C++

C++ es un lenguaje de programación de alto nivel. Es un lenguaje muy potente y tiene una comunidad de soporte muy activa por lo que está continuamente incluyendo nuevas funcionalidades y nuevas optimizaciones.

Una de las posibilidades que ofrece C++ y que resulta realmente interesante para la realización del Trabajo Fin de Grado es la capacidad de realizar códigos utilizando *templates*. Como se explicará más adelante, es necesario que puedan realizarse cálculos con una precisión a decisión del usuario de *float* o *double*. Esta necesidad es cubierta por C++ con el uso de *templates*.

Otra de las utilidades que ofrece C++ es que es un lenguaje que no necesita licencia para utilizarse.

2.4. Bibliotecas de álgebra lineal

2.4.1. Matlab

Matlab es un lenguaje de programación muy utilizado en el entorno científico gracias al amplio conjunto de herramientas que proporciona. *Matlab* cuenta con su propio entorno de desarrollo, que permite además de programar, compilar y ejecutar los códigos generados.

Cuenta con una gran comunidad de soporte y ayuda en español, que puede ser útil a la hora de entender cuál es el funcionamiento de una parte del código.

A pesar de ser una herramienta potente para realizar cálculos pesados, *Matlab* no está totalmente

optimizado para llevar a cabo tareas críticas. Esto es debido a que *Matlab* tiene una alta dependencia de su propio motor de ejecución. Por lo tanto, aunque las aplicaciones escritas en el lenguaje *Matlab* fueran ejecutadas en plataformas de altas prestaciones no se aprovecharían del todo los recursos debido a su naturaleza.

El tema principal de este Trabajo Fin de Grado es la aceleración de un procesador de imagen de resonancia magnética, y es por eso que se decide como medida inicial portar el código escrito en *Matlab* a un lenguaje de más alto nivel y más optimizado como lo es C++.

```
a = [1 2 3; 4 5 6]

b = a(:,2) * 2
```

Lista 2.1: Operaciones básicas sobre matrices.

En la Lista 2.1 se pueden ver operaciones básicas con *Matlab*. La primera operación consiste en una definición de una matriz bidimensional, la segunda operación consiste en crear un vector que corresponde con la segunda columna de a multiplicada por 2.

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad b = \begin{bmatrix} 4 \\ 10 \end{bmatrix} \tag{2.1}$$

El resultado de las operaciones de la Lista 2.1 puede verse en la Ecuación 2.1.

Desde la página principal de *Matlab* [8] se puede descargar una versión de código. Para la realización del Trabajo Fin de Grado se utilizó una licencia ofrecida por la Universidad Carlos III de Madrid.

2.4.2. GNU Octave

GNU Octave es un lenguaje de programación con una sintaxis muy similar a la de *Matlab*. Una ventaja que ofrece GNU Octave frente a *Matlab* es que es un software libre y por tanto no es necesaria una licencia para poder usarlo. Sin embargo, Octave no es tan usado en el mundo científico porque no llega a ofrecer el abanico de herramientas que *Matlab* posee.

2.4.3. Armadillo

Armadillo es una biblioteca de álgebra lineal, escrita en c++, que proporciona múltiples herramientas para realizar cálculos sobre matrices.

Una ventaja a tener en cuenta al usar esta biblioteca es que su semántica y muchas de las funciones que proporciona tienen una gran similitud con las funciones en Matlab. Esto ayudará a la hora de portar el código y a la vez para que personas que tengan experiencia con el lenguaje de Matlab puedan entender completamente el código escrito utilizando la biblioteca Armadillo.

En la página oficial de Armadillo [14] existe una sección que contiene una tabla de equivalencias de sintaxis entre *Matlab/Octave* y la sintaxis con Armadillo.

Armadillo cuenta con tipos de datos propios como *matrix*, *vector* o *cube* lo que resulta de gran utilidad en la realización del proyecto ya que se van a manipular tridimensionales para la realización de los cálculos.

En la Lista 2.2 se puede observar un código de ejemplo que está disponible en la página oficial de Armadillo. En esta lista se pueden observar operaciones tales como generación de matrices aleatorias de un tamaño elegido (mat A = randu <mat>(4,5)), multiplicado de matrices (A*B) y la operación traspuesta de una matriz (B.t()).

```
#include <iostream>
#include <armadillo>

using namespace std;
using namespace arma;

int main()
{
  mat A = randu<mat>(4,5);
  mat B = randu<mat>(4,5);

  cout << A*B.t() << endl;

  return 0;
}
```

Lista 2.2: Código ejemplo Armadillo.

2.5. Bibliotecas y lenguajes de cálculo paralelo

2.5.1. Arrayfire

Arrayfire es una biblioteca de alto rendimiento para la computación paralela. Permite realizar cálculos utilizando núcleos de CUDA y *OpenCL*. Para ello, tenemos que configurar el *backend*, que puede ser CUDA, *OpenCL* o CPU, en ese orden de preferencia.

Resulta una solución bastante acertada puesto que cuenta con sus propias llamadas para crear los núcleos necesarios para realizar los cálculos con GPUs.

En la página oficial de la biblioteca [1], tenemos las instrucciones necesarias para instalar *Arrayfire* en nuestro equipo, ya sea Windows, OSX o Linux.

2.5.2. CUDA

En la página oficial de CUDA [11] se define esta tecnología como una arquitectura de cálculo paralelo que se beneficia de las GPUs. Pertenece a la marca NVidia y solo es posible utilizarse sobre dispositivos de la misma marca.

Para poder utilizar esta tecnología es necesario descargarse los drivers de desarrollo desde la página oficial. CUDA puede ser utilizada sobre plataformas Windows, Linux y MAC.

Una de las ventajas que presenta CUDA es que puede ser implementada sobre lenguajes de alto nivel como son C, C++ o Fortran, lo que implica que sea una tecnología a tener en cuenta para nuestra solución.

2.5.3. OpenCL

Es la solución estándar de cálculo paralelo. Al contrario que CUDA es posible utilizarse sobre cualquier dispositivo. Presenta una integración con C++ lo que hace que sea una solución interesante. Al igual que CUDA esta disponible para las plataformas Windows, Linux y MAC.

2.5.4. Intel MKL

Es una biblioteca de la marca *Intel* orientada a la realización de cálculos matemáticos. Puesto que *Intel* es propietaria del software, la biblioteca está optimizada para ser usada en sistemas que cuenten con un procesador de la misma marca.

Para la realización de cálculos sobre CPU resulta muy útil, sin embargo, el propósito de este proyecto es utilizar GPUs para la aceleración. Por tanto, y a pesar de ser una solución acertada, no es la que se busca. Desde su página oficial [7], se puede descargar gratuitamente la biblioteca.

2.5.5. BLAS

Del inglés Basic Linear Algebra Subprogram. Es una biblioteca que cuenta con 3 niveles de rutinas dedicadas a realizar operaciones básicar sobre vectores y/o matrices.

- **Nivel 1:** que implementa operaciones entre números escalares, vectores y operaciones vector-vector.
- **Nivel 2:** implementa operaciones vector-matriz.
- **Nivel 3:** implementa operaciones matriz-matriz.

Desde la página oficial de la biblioteca [9], se puede descargar la última versión de código.

2.5.6. NVBlas

Es una biblioteca que implementa las rutinas de nivel 3 de BLAS. Esto es debido a que estas rutinas presentan más mejor potencial para la aceleración con GPUs. Permite integración con *Armadillo*. Para ello, es necesario añadir un fichero de configuración.

También es necesario indicar cuando se realiza la ejecución del programa que cargue la biblioteca dinámica de *NVBlas* de la siguiente forma:

```
LD_PRELOAD=/ruta\_hacia\_la\_biblioteca\_NVBlas/libnvblas.so ./phardi
```

La ruta hacia la biblioteca es dependiente del sistema en el que se ejecuta. En nuestro caso corresponde con /usr/local/cuda/lib64/.

En la página oficial de *NVBlas* [12] podemos encontrar una extensa documentación que entre otras cosas nos indica como generar el archivo de configuración para la utilización de esta biblioteca en nuestras aplicaciones.

2.5.7. LAPACK

Del inglés, Linear Algebra PACkage. Es una biblioteca escrita en FORTRAN 90 y cuenta con un amplio número de rutinas para resolver sistemas de ecuaciones lineales. Como se explica en la página oficial de LAPACK [10], el principal objetivo de esta biblioteca es el de hacer que las bibliotecas EISPACK y LINPACK ejecuten código de forma eficiente en equipos de memoria compartida y equipos con procesadores paralelos. EISPACK y LINPACK resultan ineficientes en estos equipos debido a la forma que tienen de acceder a los datos en memoria. Las rutinas de LAPACK están escritas de tal forma que cuando se realice un cálculo, éste será realizado por la biblioteca Basic Linear Algebra Subprogram (BLAS). LAPACK está desarrollado de forma que aprovecha el Nivel 3 de BLAS, nivel dedicado a las operaciones matriz-matriz.

Cuenta con una implementación en C++ llamada lapack++, lo que lleva a tener en cuenta esta tecnología.

Capítulo 3

Análisis del sistema

El objetivo principal de este capítulo es el de presentar los requisitos del proyecto que serán necesarios para la realización de los capítulos de diseño 4, e implementación 5. Para la obtención de estos requisitos, se realizaron varias reuniones en las cuáles, el tutor del proyecto presentaba las necesidades requeridas al autor del proyecto.

Una vez estudiadas dichas necesidades, el autor presentaba una lista de requisitos que debían ser aceptados o rechazados en función de si eran o no suficientes para representar las funcionalidades del sistema. Cuando todos los requisitos fueron aceptados y todas las funcionalidades cubiertas se procedió a realizar la definición del sistema, la definición de los requisitos y casos de uso.

3.1. Definición del sistema

El sistema es un programa escrito íntegramente en el lenguaje de programación C++. Este programa permite realizar operaciones sobre matrices que representan imágenes hechas por resonancia magnética de encéfalos.

Las operaciones consisten en calcular, en base a una máscara que también es proporcionada por parámetros, como de conectadas están las fibras del encéfalo. El estudio de estas conexiones permitirá el diagnóstico de enfermedades mentales como el trastorno bipolar o la esquizofrenia, Alzheimer, entre otras. Para ello, se utiliza la polarización de las moléculas de agua presentes en las fibras del cerebro.

El esquema de ejecución que sigue el sistema se puede observar en la Figura 3-1.

Figura 3-1: Descripción del sistema.

Siguiendo el esquema de la Figura 3-1 de izquierda hacia derecha, en la parte izquierda tenemos al usuario, que utilizando una terminal de comandos, elegiría los ficheros de datos, y los parámetros de ejecución elegidos. De modo que el usuario tendría que ejecutar un comando como el siguiente:

```
phardi -a rumba -k /data/data.nii.gz -m /data/nodif_brain_mask.nii.gz
-r /data/bvecs -b /data/bvals --odf /result/
```

Lista 3.1: Código ejemplo Armadillo.

En la Lista 3.1 tenemos los siguientes elementos:

- **phardi:** que es el nombre del ejecutable generado, a partir del cuál lanzaremos el programa.
- **-a:** indica el algoritmo o método que se va a utilizar para la ejecución. En nuestro caso podría ser *rumba* o *dsi*.
- **-k:** indica el fichero que contiene los datos de la imagen del encéfalo. Este fichero es una imagen en 4D. Contiene además de las 3 dimensiones necesarias para representar el cerebro, tomas realizadas en distintos ángulos.
- **-m:** indica el fichero que contiene la máscara. Es un fichero de 3 dimensiones. Este fichero indica al programa que parte del cerebro es sobre la que se quieren realizar los cálculos. Las zonas que quieren ser calculadas del cerebro están a 1 y las que no a 0.
- **-r:** indica la ruta del fichero bvecs.
- **-b:** indica la ruta del fichero bvals.
- **–odf:** indica la ruta a una carpeta existente, donde se quiere que se guarden los ficheros resultado de la ejecución.

Una vez ejecutado el comando, el programa pasaría a ejecutar el código. Dependiendo del código que se vaya a ejecutar se podrá utilizar la CPU o la GPU. En nuestro caso se han utilizado los procesadores, o si se trata de operaciones con matrices, entonces esos cálculos serán derivados a la GPU, en nuestro proyecto utilizamos tarjetas gráficas de los modelos GeForce GTX TITAN Black.

Cuando todos los cálculos están realizados, el programa guarda los resultados en ficheros en la ruta que se eligió mediante el parámetro –odf. Esa ruta hacer referencia a una carpeta del sistema que tiene que existir.

3.2. Casos de uso

En este apartado se presentan las funcionalidades que se quieren implementar en el sistema.

3.2.1. Definición de los casos de uso

Para representar cada caso de uso que el usuario puede realizar sobre el sistema se utilizará una tabla como la siguiente:

Tabla 3.1: Tabla ejemplo de los Casos de Uso.

Identificador
Nombre
Descripción
Precondiciones
Postcondiciones

Tal y como se muestra en la Tabla 3.1, cada caso de uso contará con los siguientes campos:

- **Identificador:** Código unívoco que sirve para identificar cada uno de los casos de uso. El formato de éste código es el siguiente:
  - **CU:** Las primeras dos letras indican que se trata de un caso de uso.
  - **XX:** las dos X siguientes representan la numeración del caso de uso en referencia con el número total.

- **Nombre:** Título a modo de breve resumen de la descripción del caso de uso.
- **Descripción:** Explicación detallada del caso de uso.
- **Precondiciones:** Lista de condiciones necesarias para que el caso de uso pueda llevarse a cabo.
- **Postcondiciones:** Consecuencias que conlleva la realización del caso de uso.

Tabla 3.2: Tabla Caso de Uso CU-01.

Identificador	CU-01
Nombre	Uso básico de la aplicación
Descripción	El usuario utiliza la aplicación.
Precondiciones	▪ El usuario debe escribir en la terminal de comandos "./phardi -a algoritmo -k /path_to_data/data.nii.gz -m /path_to_data/nodif_brain_mask.nii.gz -r /path_to_data/bvecs -b /path_to_data/bvals –odf /result/" ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Se guardan los ficheros de resultado en la carpeta indicada.

Tabla 3.3: Tabla Caso de Uso CU-02.

Identificador	CU-02
Nombre	Utilizar método de reconstrucción *rumba*
Descripción	El usuario elige como método de reconstrucción el algoritmo *rumba*.
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Se guardan los ficheros de resultado en la carpeta indicada.

Tabla 3.4: Tabla Caso de Uso CU-03.

Identificador	CU-03
Nombre	Utilizar método de reconstrucción *dsi*
Descripción	El usuario elige como método de reconstrucción el algoritmo *dsi*
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Se guardan los ficheros de resultado en la carpeta indicada.

Tabla 3.5: Tabla Caso de Uso CU-04.

Identificador	CU-04
Nombre	Comprimir los ficheros resultantes
Descripción	El usuario elige comprimir los ficheros resultantes del cálculo
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes ▪ Como argumento adicional debe introducirse la opción de comprimir, es decir, “-compress” o “-z”
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Se guardan los ficheros de resultado en la carpeta indicada. ▪ Se comprimen los ficheros.

Tabla 3.6: Tabla Caso de Uso CU-05.

Identificador	CU-05
Nombre	Utilizar precisión simple para realizar los cálculos
Descripción	El usuario elige realizar los cálculos con una precisión simple lo que supone utilizar un tipo de datos denominado *float*
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes ▪ Como argumento adicional debe introducirse la opción de precisión, es decir, "–precision" o "-p" seguido de "float".
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Para realizar los cálculos se utilizan datos de precisión simple. ▪ Se guardan los ficheros de resultado en la carpeta indicada.

Tabla 3.7: Tabla Caso de Uso CU-06.

Identificador	CU-06
Nombre	Utilizar precisión doble para realizar los cálculos
Descripción	El usuario elige realizar los cálculos con una precisión simple lo que supone utilizar un tipo de datos denominado *double*
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes ▪ Como argumento adicional debe introducirse la opción de precisión, es decir, "–precision" o "-p" seguido de "double".
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Para realizar los cálculos se utilizan datos de precisión doble. ▪ Se guardan los ficheros de resultado en la carpeta indicada.

Tabla 3.8: Tabla Caso de Uso CU-07.

Identificador	CU-07
Nombre	Visualizar uso de la aplicación
Descripción	El usuario desea ver cómo debe lanzarse la aplicación
Precondiciones	▪ El usuario debe escribir por línea de comandos "./phardi –help" ó "./phardi -h"
Postcondiciones	Se imprime por la terminal de comandos cómo debe lanzarse la aplicación y termina la ejecución.

Tabla 3.9: Tabla Caso de Uso CU-08.

Identificador	CU-08
Nombre	El usuario utiliza el método de lectura de datos *slice*
Descripción	El usuario elige que para la lectura de los datos se utilice el método *slice*
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes ▪ Como argumento adicional debe introducirse la opción de lectura de datos, es decir, "–dataread" o "-d" seguido de "slice".
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Para realizar la lectura de los datos se utiliza el método *slice*. ▪ Se guardan los ficheros de resultado en la carpeta indicada.

Tabla 3.10: Tabla Caso de Uso CU-09.

Identificador	CU-09
Nombre	El usuario utiliza el método de lectura de datos *volume*
Descripción	El usuario elige que para la lectura de los datos se utilice el método *volume*
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes ▪ Como argumento adicional debe introducirse la opción de lectura de datos, es decir, "–dataread" o "-d" seguido de "volume".
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Para realizar la lectura de los datos se utiliza el método *volume*. ▪ Se guardan los ficheros de resultado en la carpeta indicada.

Tabla 3.11: Tabla Caso de Uso CU-10.

Identificador	CU-10
Nombre	El usuario utiliza el método de lectura de datos *voxel*
Descripción	El usuario elige que para la lectura de los datos se utilice el método *voxel*
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes ▪ Como argumento adicional debe introducirse la opción de lectura de datos, es decir, "–dataread" o "-d" seguido de "voxel".
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Para realizar la lectura de los datos se utiliza el método *voxel*. ▪ Se guardan los ficheros de resultado en la carpeta indicada.

Tabla 3.12: Tabla Caso de Uso CU-11.

Identificador	CU-11
Nombre	El usuario utiliza la aplicación en modo *verbose*
Descripción	El usuario elige ejecutar la aplicación con el modo *verbose*, en el cuál se van mostrando datos sobre la ejecución.
Precondiciones	▪ Los argumentos deben ser introducidos de forma correcta ▪ Debe elegirse una carpeta existente para guardar los ficheros resultantes ▪ Como argumento adicional debe introducirse la opción *verbose*, es decir, "–verbose" o "-v".
Postcondiciones	▪ Se ejecuta la aplicación utilizando el algoritmo y los datos especificados. ▪ Durante la ejecución del programa se muestra información.

3.3. Definición de requisitos

En este apartado del documento se va a exponer el conjunto de requisitos del sistema. Estos requisitos se dividen en tres grandes grupos, requisitos funcionales, requisitos no funcionales y requisitos de software.

3.3.1. Requisitos funcionales

Los requisitos funcionales hacen referencia a las capacidades o funcionalidades que debe presentar el sistema. Para representar cada requisito funcional se utilizará la siguiente tabla:

Identificador	RF-XX
Nombre	Nombre del requisito
Fuente	Fuente del requisito
Prioridad	☐ Alta ☐ Media ☐ Baja
Verificabilidad	☐ Alta ☐ Media ☐ Baja
Descripción	Descripción del requisito

Tabla 3.13: Tabla ejemplo de requisitos funcionales

Como vemos en la tabla 3.13, para cada requisito tenemos las siguientes características:

- **Identificador:** Código unívoco que sirve para identificar cada uno de los requisitos. El formato de este código es el siguiente:
  - **RF:** Las primeras dos letras indican que se trata de un requisito funcional.
  - **XX:** las dos X siguientes representan la numeración del requisito en referencia con el número total de requisitos de ese tipo.
- **Nombre:** Título a modo de breve resumen de la descripción del requisito.
- **Prioridad:** Indica el nivel de necesidad con el que debe realizarse el requisito.
- **Verificabilidad:** Capacidad del requisito para comprobar su funcionamiento.
- **Descripción:** Explicación detallada del requisito.

Tabla 3.14: Requisito funcional RF-01.

Identificador	RF-01
Nombre	La aplicación debe poder ejecutarse con el método *rumba*
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☐ Alta ☑ Media ☐ Baja
Descripción	El código desarrollado debe permitir ejecuciones con el método de reconstrucción *rumba*

Tabla 3.15: Requisito funcional RF-02.

Identificador	RF-02
Nombre	La aplicación debe poder ejecutarse con el método *dsi*
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☐ Alta ☑ Media ☐ Baja
Descripción	El código desarrollado debe permitir ejecuciones con el método de reconstrucción *dsi*

Tabla 3.16: Requisito funcional RF-03.

Identificador	RF-03
Nombre	El sistema debe soportar imágenes en el formato *NIfTI*
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	El sistema tiene que poder leer las imágenes del formato *NIfTI*, que actualmente lee el sistema escrito en *Matlab*

Tabla 3.17: Requisito funcional RF-04.

Identificador	RF-04
Nombre	El sistema debe poder instalarse en sistemas Linux
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación final debe poder instalarse en máquinas con sistemas operativos basados el Linux

Tabla 3.18: Requisito funcional RF-05.

Identificador	RF-05
Nombre	El sistema debe poder instalarse en sistemas Windows
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación final debe poder instalarse en máquinas con el sistema operativo Microsoft Windows

Tabla 3.19: Requisito funcional RF-06.

Identificador	RF-06
Nombre	El sistema debe soportar ejecución sobre GPUs
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación final debe soportar la ejecución de cálculos de álgebra lineal sobre GPUs

Tabla 3.20: Requisito funcional RF-07.

Identificador	RF-07
Nombre	El sistema debe permitir método de lectura *slice*.
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación debe permitir que el método de lectura de los dato de los cerebros sea *slice*

Tabla 3.21: Requisito funcional RF-08.

Identificador	RF-08
Nombre	El sistema debe permitir método de lectura *volume*.
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación debe permitir que el método de lectura de los dato de los cerebros sea *volume*

Tabla 3.22: Requisito funcional RF-09.

Identificador	RF-09
Nombre	El sistema debe permitir método de lectura *voxel*.
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación debe permitir que el método de lectura de los dato de los cerebros sea *voxel*

Tabla 3.23: Requisito funcional RF-10.

Identificador	RF-10
Nombre	El sistema debe permitir la ejecución con datos de precisión simple ó *float*.
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación debe permitir que los cálculos se realicen con una precisión simple, es decir, que se utilice un tipo de dato denominado *float*.

Tabla 3.24: Requisito funcional RF-11.

Identificador	RF-11
Nombre	El sistema debe permitir la ejecución con datos de precisión doble ó *double*.
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación debe permitir que los cálculos se realicen con una precisión simple, es decir, que se utilice un tipo de dato denominado *double*.

Tabla 3.25: Requisito funcional RF-12.

Identificador	RF-12
Nombre	El sistema debe contar con una opción de ayuda.
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación debe permitir que mediante una opción, se le indique al usuario cómo debe lanzar la aplicación y que parámetros son requeridos.

Tabla 3.26: Requisito funcional RF-13.

Identificador	RF-13
Nombre	El sistema debe permitir comprimir los ficheros resultantes.
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación final debe permitir al usuario que los ficheros resultantes de la ejecución sean comprimidos y guardados en la carpeta que se indicó por parámetros.

Tabla 3.27: Requisito funcional RF-14.

Identificador	RF-14
Nombre	El sistema debe permitir la opción *verbose*.
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación final debe permitir al usuario la utilización de la opción *verbose*. Esto hará que durante la ejecución de la aplicación, se muestre información de la misma.

3.3.2. Requisitos no funcionales

El último tipo de requisitos hace referencia a la parte no funcional del trabajo, es decir, rendimiento, seguridad. La tabla que se va a utilizar para expresar este tipo de requisitos es similar a la utilizada para los requisitos funcionales:

Tabla 3.28: Tabla ejemplo de requisitos no funcionales.

Identificador	RNFXX
Nombre	Nombre del requisito
Fuente	Tutor
Prioridad	☐ Alta ☐ Media ☐ Baja
Verificabilidad	☐ Alta ☐ Media ☐ Baja
Descripción	Descripción detallada del requisito

Al igual que en la tabla de requisitos funcionales tenemos las siguientes características:

- **Identificador:** Código unívoco que sirve para identificar cada uno de los requisitos. El formato de éste código es el siguiente:
  - **RNF:** Las primeras dos letras indican que se trata de un requisito no funcional.
  - **XX:** las dos X siguientes representan la numeración del requisito en referencia con el número total de requisitos de ese tipo.
- **Nombre:** Título a modo de breve resumen de la descripción del requisito.
- **Prioridad:** Indica el nivel de necesidad con el que debe realizarse el requisito.
- **Verificabilidad:** Capacidad del requisito para comprobar su funcionamiento.
- **Descripción:** Explicación detallada del requisito.

Tabla 3.29: Requisito no funcional RNF-01.

Identificador	RNF-01
Nombre	El sistema debe mejorar los tiempos de ejecución
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	El código desarrollado debe mejorar los tiempos de ejecución de la aplicación anterior.

Tabla 3.30: Requisito no funcional RNF-02.

Identificador	RNF-02
Nombre	Código fácilmente identificable
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	El código generado en C++ debe ser fácilmente identificable por usuarios de MATLAB

Tabla 3.31: Requisito no funcional RNF03.

Identificador	RNF-03
Nombre	Código organizado
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	Los ficheros de código generados deben estar ordenados por carpetas con nombres significativos.

Tabla 3.32: Requisito no funcional RNF-04.

Identificador	RNF-04
Nombre	El sistema debe generar imágenes iguales a la que se generaban con la aplicación anterior
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación final debe generar imágenes iguales a las que generaba la aplicación anterior.

3.3.3. Requisitos software

Estos son los requisitos que hacen referencia a las especificaciones del software. Se utilizará una tabla equivalente a las tablas de requisitos funcionales y no funcionales:

Tabla 3.33: Tabla ejemplo de requisitos software.

Identificador	RSXX
Nombre	Nombre del requisito
Fuente	Tutor
Prioridad	☐ Alta ☐ Media ☐ Baja
Verificabilidad	☐ Alta ☐ Media ☐ Baja
Descripción	Descripción detallada del requisito

Como en las tablas anteriores, contamos con los siguientes campos:

- **Identificador:** Código unívoco que sirve para identificar cada uno de los requisitos. El formato de éste código es el siguiente:
  - **RS:** Las primeras dos letras indican que se trata de un requisito software.
  - **XX:** las dos X siguientes representan la numeración del requisito en referencia con el número total de requisitos de ese tipo.
- **Nombre:** Título a modo de breve resumen de la descripción del requisito.
- **Prioridad:** Indica el nivel de necesidad con el que debe realizarse el requisito.
- **Verificabilidad:** Capacidad del requisito para comprobar su funcionamiento.
- **Descripción:** Explicación detallada del requisito.

Tabla 3.34: Requisito funcional RS-01.

Identificador	RS-01
Nombre	Aplicación escrita en C++
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación debe estar escrita íntegramente en el lenguaje de programación C++

Tabla 3.35: Requisito funcional RS-02.

Identificador	RS-02
Nombre	El código debe poder soportar OpenMP
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☐ Alta ☑ Media ☐ Baja
Descripción	El código desarrollado debe permitir sentencias OpenMP que permitan ejecutar código en paralelo

Tabla 3.36: Requisito funcional RS-03.

Identificador	RS-03
Nombre	La aplicación final debe soportar la integración con la biblioteca Armadillo
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación desarrollada tiene que contener sentencias de la biblioteca Armadillo, para las operaciones de álgebra lineal

Tabla 3.37: Requisito funcional RS-04.

Identificador	RS-04
Nombre	La aplicación final debe soportar la integración con la biblioteca NVBlas
Fuente	Tutor
Prioridad	☑ Alta ☐ Media ☐ Baja
Verificabilidad	☑ Alta ☐ Media ☐ Baja
Descripción	La aplicación desarrollada debe permitir la integración con la biblioteca NVBlas que permita la ejecución de operaciones de álgebra lineal sobre GPUs

3.4. Matrices de trazabilidad

3.4.1. Requisitos-Casos de Uso

La Tabla 3.38 muestra una matriz de trazabilidad entre requisitos funcionales y casos de uso. De esta forma se comprueba que todos los casos de uso están cubiertos por los requisitos.

Tabla 3.38: Matriz trazabilidad casos de uso requisitos.

	CU-01	CU-02	CU-03	CU-04	CU-05	CU-06	CU-07	CU-08	CU-09	CU-10	CU-11
RF-01	✓	✓	-	-	-	-	-	-	-	-	-
RF-02	✓	-	✓	-	-	-	-	-	-	-	-
RF-03	✓	-	-	-	-	-	-	-	-	-	-
RF-04	-	-	-	-	-	-	-	-	-	-	-
RF-05	-	-	-	-	-	-	-	-	-	-	-
RF-06	-	-	-	-	-	-	-	-	-	-	-
RF-07	✓	-	-	-	-	-	-	✓	-	-	-
RF-08	-	-	-	-	-	-	-	-	✓	-	-
RF-09	-	-	-	-	-	-	-	-	-	✓	-
RF-10	✓	-	-	-	✓	-	-	-	-	-	-
RF-11	-	-	-	-	-	✓	-	-	-	-	-
RF-12	-	-	-	-	-	-	✓	-	-	-	-
RF-13	-	-	-	✓	-	-	-	-	-	-	-
RF-14	-	-	-	-	-	-	-	-	-	-	✓

3.4.2. Requisitos-Resquisitos

La Tabla 3.39 muestra una matriz de trazabilidad entre requisitos. Esta matriz nos permite detectar la relación de los requisitos entre sí.

Tabla 3.39: Matriz trazabilidad requisitos - requisitos.

	RF-01	RF-02	RF-03	RF-04	RF-05	RF-06	RF-07	RF-08	RF-09	RF-10	RF-11	RF-12	RF-13	RF-14	RNF-01	RNF-02	RNF-03	RNF-04
RF-01		-	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	-	-	-	-
RF-02	-		✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	-	-	-	-
RF-03	✓	✓		-	-	-	✓	✓	✓	-	-	-	✓	-	-	-	-	-
RF-04	✓	✓	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-
RF-05	✓	✓	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-
RF-06	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-
RF-07	✓	✓	✓	-	-	-		-	-	-	-	-	-	-	-	-	-	-
RF-08	✓	✓	✓	-	-	-	-		-	-	-	-	-	-	-	-	-	-
RF-09	✓	✓	✓	-	-	-	-	-		-	-	-	-	-	-	-	-	-
RF-10	✓	✓	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-
RF-11	✓	✓	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-
RF-12	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-
RF-13	✓	✓	✓	-	-	-	-	-	-	-	-	-		-	-	-	-	-
RF-14	✓	✓	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-
RNF-01	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-
RNF-02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-
RNF-03	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-
RNF-04	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

Capítulo 4

Diseño

El propósito de este capítulo es explicar de forma detallada los distintos componentes que intervienen en la aplicación final. En este capítulo también se recogerá el diagrama de clases referente a la aplicación y una breve explicación de los argumentos recibidos por terminal de comandos.

4.1. Diseño de clases

En la Figura 4-1 se puede observar el diagrama de clases. Como tal los ficheros no deben ser definidos como clases ya que se trata de ficheros con extensión *.hpp*. Esto indica que son ficheros de cabecera o, lo que es lo mismo, un fichero de declaración. Se utilizan como ficheros de cabecera debido a que las funciones son definidas como *templates*. La definición de funciones *templates* es útil en esta aplicación ya que se quiere dar la posibilidad al usuario de poder utilizar precisión simple y doble. Dependiendo del algoritmo que elija el usuario, el programa tomará una u otra línea de ejecución.

Tal y cómo muestra la Figura 4-1, la clase *phardi* es la que contiene el método *main* y por tanto la que interactuará con el usuario. En esta clase es donde se procesarán los parámetros de entrada. Los argumentos que se reciben son los siguientes:

- **data:** se trata de un conjunto de datos 4D en formato *Nifti*.
- **nodif_brain_mask:** se trata de una máscara binaria en 3D que indica que zonas del cerebro son las que se quieren procesar.

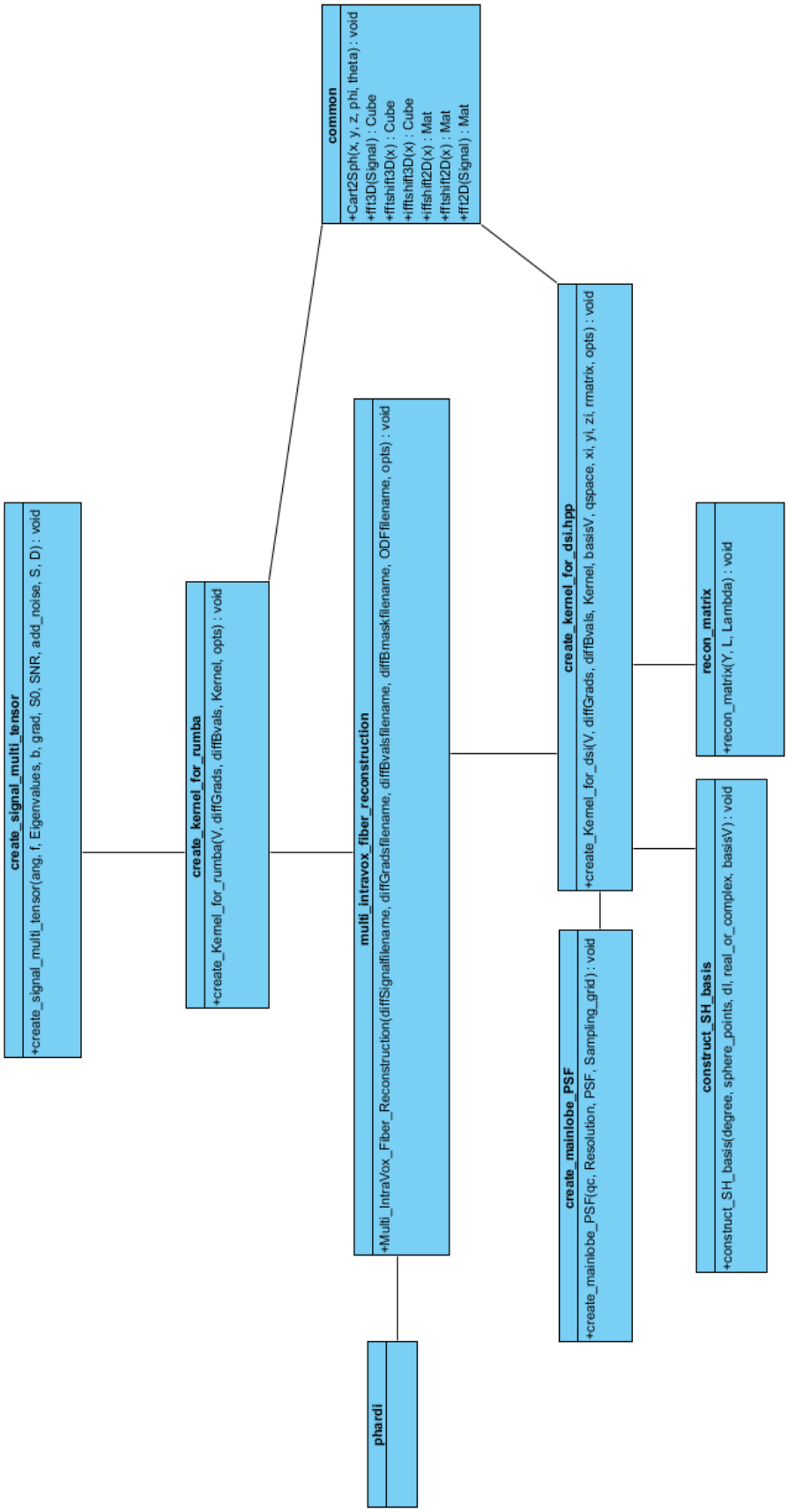

Figura 4-1: Diagrama de clases.

$$\begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ y_1 & y_2 & y_3 & \dots & y_n \\ z_1 & z_2 & z_3 & \dots & z_n \end{bmatrix}$$

Figura 4-2: Formato de los datos en el archivo bvecs

$$b_1 \quad b_2 \quad b_3 \quad \dots \quad b_n$$

Figura 4-3: Formato de los datos en el archivo bvals.

- **bvecs:** un archivo de texto ASCII que contiene una lista de gradientes que son aplicados durante la difusión ponderada de volúmenes. El orden de los datos de este archivo debe coincidir con el orden de los volúmenes en los datos. El formato de este fichero puede observarse en la Figura 4-2.
- **bvals:** es un fichero de texto que contiene los *bvalues* aplicados durante cada adquisición de volumen. El orden de las entradas en este fichero debe coincidir con el orden de los volúmenes en el fichero de datos y con las entradas del fichero de gradiente (Figura 4-3).
- **carpeta destino:** que es una carpeta existente del equipo donde se ejecuta la aplicación. En ella se guardarán los archivos resultantes.

La clase *multi intravox* es la encargada de realizar los cálculos dependiendo del algoritmo que elija el usuario y del método de lectura de los datos. El método de lectura de los datos estará por defecto en *slices* y solo se actualizará en caso de que se indique en la ejecución. Dependiendo del algoritmo elegido se llamará a la clase que genera el kernel de *rumba* o de *dsi*. La clase *common* contendrá funciones o métodos que sean de uso general por la aplicación.

Capítulo 5

Implementación

El principal objetivo de este capítulo es el de explicar la implementación de las clases que se han definido en el Capítulo de diseño 4. Por cada clase, se explicará los parámetros de entrada, el resultado que devuelve, y una breve descripción de la funcionalidad. Para algunas de estas clases también se ha estimado oportuno incluir parte del código.

5.1. phardi.cpp

Es la clase encargada de iniciar el programa. En ella se encuentra la función *main* y, por tanto, es la encargada de procesar los argumentos que introduce el usuario.

Una vez procesados los argumentos, se pasan a una estructura de opciones. La estructura tiene la siguiente forma:

```
struct options {
    // Reconstruction method.
    recons         reconsMethod;
    datread        datreadMethod;
    // Data reading method.
    std::string    outputDir;
    std::string    ODFDirscheme;
    // Directions scheme for reconstructing ODF.
    options_rumba  rumba_sd;
    options_dsi    dsi;
    bool           zip;
```

```
    bool            debug;
};
```

Lista 5.1: Código de estructura de opciones.

La primera variable indica el método de reconstrucción que en este caso sería *rumba* o *dsi*. La siguiente variable indicaría el método en el que van a ser leídos los datos que puede ser: *slice*, *volume* o *voxel*. Las variables *rumba* y *dsi* hacen referencia a lo parámetros de cada uno de los métodos. Cuando se han guardado todas las opciones en el objeto de opciones se pasa a llamar a la función *Multi_IntraVox_Fiber_Reconstruction*. Al tratarse de un *template*, hay que indicarle que tipo de datos vamos a utilizar.

5.2. Multi Intravox Fiber Reconstruction

Esta función realiza los cálculos de la distribución de las orientaciones de las fibras de todo el cerebro.

Como parámetros recibe:

- **diffImage:** variable de tipo *String*. Es la imagen que se va a procesar.
- **bvecs:** variable de tipo *String*.
- **bvals:** variable de tipo *String*.
- **diffBmask:** variable de tipo *String*. es un objeto 3D que indica que región del cerebro es la que se quiere tener en cuenta.
- **ODFfilename:** variable de tipo *String*.
- **opts:** variable del tipo *options* tal y como puede verse en el código del apartado anterior. Contiene las opciones de ejecución del programa.

La mayoría de este archivo corresponde con operaciones realizadas sobre los datos de entrada. Estas operaciones han sido traducidas directamente desde *Matlab* a C++. Dependiendo del método que se quiere utilizar se generará un núcleo distinto. Las dos posibles opciones son un núcleo de *rumba* o un núcleo de *dsi*.

Una vez creado el núcleo se pasará a comprobar el método de lectura de los datos que como ya se ha mencionado en el apartado anterior puede ser: *slice*, *volume* o *voxel*. Finalmente, crea el archivo ODF(del inglés, Orientational Distribution Function) en la ruta especificada.

5.3. Núcleo de RUMBA

Esta función crea el núcleo de *rumba*, que posteriormente será utilizado para calcular las orientaciones de las fibras del cerebro.

Recibe como parámetros:

- **V:** variable del tipo *Mat* de *Armadillo*.
- **diffGrads:** variable del tipo *Mat* de *Armadillo*.
- **diffBvals:** variable del tipo *Col* de *Armadillo*.
- **Kernel:** variable del tipo *Mat* de *Armadillo*.
- **opts:** variable del tipo *options*. Contiene las opciones de ejecución del programa.

El núcleo generado final, se guarda en la matriz *Kernel* y es devuelto en esa misma matriz ya que es pasada por referencia.

En la Lista 5.2 podemos observar el código completo de esta función. El código está escrito en C++, en él se pueden observar el uso variables y de operaciones de *Armadillo* y *pragmas* de *OpenMP*.

```
/**
* @version              pHARDI v0.3
* @copyright            Copyright (C) 2017 Universidad Carlos III de Madrid. All rights
    reserved.
* @license              GNU/GPL, see LICENSE.txt
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You have received a copy of the GNU General Public License in LICENSE.txt
* also available in <http://www.gnu.org/licenses/gpl.html>.
*
* See COPYRIGHT.txt for copyright notices and details.
*/
```

```
#ifndef CREATE_KERNEL_RUMBA_H
#define CREATE_KERNEL_RUMBA_H

#include "create_signal_multi_tensor.hpp"
#include "common.hpp"

#include <plog/Log.h>
#include <iostream>
#include <math.h>
#include <armadillo>

namespace phardi {
    template <typename T>
    void create_Kernel_for_rumba(const arma::Mat<T> & V,
                                 const arma::Mat<T> & diffGrads,
                                 const arma::Col<T> & diffBvals,
                                 arma::Mat<T> & Kernel,
                                 const phardi::options opts) {

        using namespace arma;
        // add_rician_noise = 0;
        bool add_rician_noise = opts.rumba_sd.add_noise;

        T SNR = 1;

        // [phi, theta] = cart2sph(V(:,1),V(:,2),V(:,3)); % set of directions
        Col<T> phi(V.n_rows);
        Col<T> theta(V.n_rows);

  #pragma omp parallel for
        for (int i = 0; i < V.n_rows; ++i) {
            Cart2Sph(V(i,0),V(i,1),V(i,2),phi(i),theta(i));
        }

      theta.transform( [](T val) { return (-val); } );

        //S0 = 1; % The Dictionary is created under the assumption S0 = 1;
        T S0 = 1;
        //fi = 1; % volume fraction
        Col<T> fi(1);
        fi(0) = 1;

        Col<T> S(diffGrads.n_rows);
        Mat<T> D(3,3);
```

```
    Mat<T> v2(fi.n_elem,2);
    Col<T> v3(3);

    T c = 180/M_PI;

    //for i=1:length(phi)
    for (size_t i = 0; i < phi.n_elem; i++) {
        // anglesFi = [phi(i), theta(i)]*(180/pi); %in degrees
        v2(0,0) = phi(i)*c; v2(0,1) = theta(i)*c;
        v3(0) = opts.rumba_sd.lambda1; v3(1) = opts.rumba_sd.lambda2;  v3(2) = opts.
rumba_sd.lambda2;

        S.fill(0.0);
        // Kernel(:,i) = create_signal_multi_tensor(anglesFi, fi, [lambda1, lambda2,
lambda2], diffBvals, diffGrads, S0, SNR, add_rician_noise);
        create_signal_multi_tensor<T>(v2, fi, v3, diffBvals, diffGrads, S0, SNR,
add_rician_noise, S, D);
        for (size_t j = 0; j<S.n_elem;++j) {
            Kernel(i,j) = S(j);
        }
    }

    S.fill(0.0);
    v2(0,0) = phi(0)*c; v2(0,1) = theta(0)*c;
    v3(0) = opts.rumba_sd.lambda_csf; v3(1) = opts.rumba_sd.lambda_csf;  v3(2) = opts.
rumba_sd.lambda_csf;
    create_signal_multi_tensor<T>(v2, fi, v3, diffBvals, diffGrads, S0, SNR,
add_rician_noise, S, D);

  #pragma omp parallel for
    for (size_t i = 0; i<S.n_elem; ++i) {
        Kernel(phi.n_elem  ,i) = S(i);
    }

    S.fill(0.0);
    v2(0,0) = phi(0)*c; v2(0,1) = theta(0)*c;
    v3(0) = opts.rumba_sd.lambda_gm; v3(1) = opts.rumba_sd.lambda_gm;  v3(2) = opts.
rumba_sd.lambda_gm;
    create_signal_multi_tensor<T>(v2, fi, v3, diffBvals, diffGrads, S0, SNR,
add_rician_noise, S, D);

  #pragma omp parallel for
    for (size_t i = 0; i<S.n_elem; ++i) {
        Kernel(phi.n_elem + 1 ,i) = S(i);
    }
```

```
        Kernel = Kernel.t();
    }
}
#endif
```

Lista 5.2: Código Núcleo *RUMBA*.

5.3.1. Create Signal Multi Tensor

Como parámetros recibe:

- **diffImage:** variable de tipo *String*. Es la imagen que se va a procesar
- **bvecs**: variable de tipo *String*.
- **bvals:** variable de tipo *String*.
- **diffBmask:** variable de tipo *String*. es un objeto 3D que indica que región del cerebro es la que se quiere tener en cuenta.
- **ODFfilename:** variable de tipo *String*.
- **opts:** variable del tipo *options* tal y como puede verse en el código del apartado anterior. Contiene las opciones de ejecución del programa.

5.4. Núcleo de DSI

Esta función crea el núcleo de *dsi*, que posteriormente será utilizado para calcular las orientaciones de las fibras del cerebro.

Recibe como parámetros:

- **V:** variable del tipo *Mat* de *Armadillo*.
- **diffGrads:** variable del tipo *Mat* de *Armadillo*.
- **diffBvals:** variable del tipo *Col* de *Armadillo*.
- **Kernel:** variable del tipo *Mat* de *Armadillo*.

- **basisV:** variable del tipo *Mat* de *Armadillo*.
- **qspace:** variable del tipo *Mat* de *Armadillo*.
- **xi:** variable del tipo *Mat* de *Armadillo*.
- **yi:** variable del tipo *Mat* de *Armadillo*.
- **zi:** variable del tipo *Mat* de *Armadillo*.
- **rmatrix:** variable del tipo *Mat* de *Armadillo*.
- **opts:** variable del tipo *options*. Contiene las opciones de ejecución del programa.

En esta función primero se calcula cuál sería el centro de la imagen. Una vez calculado se pasa a inicializar las variables *xi*, *yi* y *zi* de la siguiente forma:

```
for(uword m = 1; m < V.n_rows; m++){
    xi.row(m) = center_of_image + r * V(m,1);
    yi.row(m) = center_of_image + r * V(m,0);
    zi.row(m) = center_of_image + r * V(m,2);
}
```

Lista 5.3: Código inicializado variables *xi yi* y *zi*.

Es decir, cada una de estas matrices, xi, yi y zi se añaden las filas correspondientes de la matriz V. Después de realizar estos cálculos se llama a la función *Create mainlobe PSF*. Cuando tenemos el cubo PSF calculado se pasa a crear lo que llama en el programa Spherical Harmonics.

Ahora se pasa a calcular la matriz basisv a partir de la función *contruct_SH_basis*. Como paso final se hace una llamada a la función recon_matrix cuyo resultado se devolverá en la variable *Kernel* y terminará la ejecución de la función de creación de núcleo para el método *dsi*.

En la Lista 5.4 podemos observar el código completo de esta función. El código está escrito en C++, en él se pueden observar el uso variables y de operaciones de *Armadillo* y *pragmas* de *OpenMP*.

```
/**
* @version              pHARDI v0.3
* @copyright            Copyright (C) 2017 Universidad Carlos III de Madrid. All rights
    reserved.
* @license              GNU/GPL, see LICENSE.txt
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
```

```
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You have received a copy of the GNU General Public License in LICENSE.txt
* also available in <http://www.gnu.org/licenses/gpl.html>.
*
* See COPYRIGHT.txt for copyright notices and details.
*/

#ifndef CREATE_KERNEL_DSI_H
#define CREATE_KERNEL_DSI_H

#define __STDCPP_WANT_MATH_SPEC_FUNCS__ 1


#include "common.hpp"

#include <plog/Log.h>
#include <iostream>
#include <math.h>
#include <cmath>
#include <armadillo>

namespace phardi {

    template <typename T>
    void create_Kernel_for_dsi(const arma::Mat<T> & V,
                               const arma::Mat<T> & diffGrads,
                               const arma::Col<T> & diffBvals,
                               arma::Mat<T> & Kernel,
                               arma::Mat<T> & basisV,
                               arma::Mat<arma::uword> & qspace,
                               arma::Mat<T> & xi,
                               arma::Mat<T> & yi,
                               arma::Mat<T> & zi,
                               arma::Mat<T> &rmatrix,
                               const phardi::options opts) {

        using namespace arma;

        Cube<T> PSF;
```

```
        Cube<T> Sampling_grid;

        Col<T> phi;
        Col<T> theta;
        Col<uword> Laplac2;
        std::vector<uword> Laplac2_v;

        // center_of_image = (opts.dsi.resolution-1)/2 + 1;
        uword center_of_image = (opts.dsi.resolution - 1)/2 ;

        //rmin = opts.dsi.rmin;
        uword rmin = opts.dsi.rmin;

        //rmax = opts.dsi.resolution - center_of_image - 1;
        uword rmax = opts.dsi.resolution - center_of_image - 2;

        //r = rmin:(rmax/100):rmax; % radial points that will be used for the ODF radial
    summation
        Row<T> r = regspace<Row<T>>(rmin, rmax/100.0, rmax);

        //rmatrix = repmat(r,[length(V), 1]);
        rmatrix = repmat(r, V.n_rows, 1);

        // for m=1:length(V)
        //     xi(m,:) = center_of_image + r*V(m,2);
        //     yi(m,:) = center_of_image + r*V(m,1);
        //     zi(m,:) = center_of_image + r*V(m,3);
        // end

        xi.resize(V.n_rows,r.n_elem);
        yi.resize(V.n_rows,r.n_elem);
        zi.resize(V.n_rows,r.n_elem);

#pragma omp parallel for
        for (uword m = 0; m < V.n_rows; ++m) {
            xi.row(m) = center_of_image + 1 + r * V(m, 1);
            yi.row(m) = center_of_image + 1 + r * V(m, 0);
            zi.row(m) = center_of_image + 1 + r * V(m, 2);
        }

        // --- q-space points centered in the new matrix of dimensions Resolution x
    Resolution x Resolution
        // grad_qspace = round(opts.dsi.boxhalfwidth*diffGrads.*sqrt([diffBvals diffBvals
    diffBvals]/max(diffBvals)));
        Mat<T> grad_qspace = round(opts.dsi.boxhalfwidth * diffGrads % sqrt(join_rows(
```

```
    join_rows(diffBvals, diffBvals), diffBvals) / max(diffBvals)));

        // qspace = grad_qspace + center_of_image;
        qspace = conv_to<Mat<uword>>::from(grad_qspace + center_of_image) ;

        // - Computing the point spread function (PSF) for the deconvolution operation
        //   ------------------------------------------------------

        // [PSF, Sampling_grid] = create_mainlobe_PSF(qspace,opts.dsi.resolution);
        create_mainlobe_PSF(qspace, opts.dsi.resolution, PSF, Sampling_grid);

        // Creating Spherical Harmonics
        // Laplac2 = [];
        // for L=0:2:opts.dsi.lmax
        //     for m=-L:L
                //Laplac2 = [Laplac2; (L^2)*(L + 1)^2];
            //end
        // end

        Laplac2.reset() ;
        for (sword L=0; L <= opts.dsi.lmax; L+=2)
        {
            for (sword m=-(L); m <= L; ++m)
            {
                Laplac2_v.push_back((pow(L, 2))*pow((L+1), 2));
            }
        }
        Laplac2.set_size(Laplac2_v.size());
        Laplac2 = conv_to<Col<uword>>::from(Laplac2_v);
        //Laplac = diag(Laplac2);
        Mat<uword> Laplac = diagmat(Laplac2);
        // basisV = construct_SH_basis (opts.dsi.lmax, V, 2, 'real');
        construct_SH_basis<T>(opts.dsi.lmax, V, 2, "real", theta, phi, basisV) ;
        // Kernel = recon_matrix(basisV,Laplac,opts.dsi.lreg);
        Kernel = recon_matrix<T>(basisV, Laplac, opts.dsi.lreg);
        return;
    }
}
#endif
```

Lista 5.4: Código Núcleo *DSI*.

5.4.1. Create mainlobe PSF

Recibe como parámetros:

- **qc:** variable del tipo *Mat* de *Armadillo*.
- **Resolution:** variable del tipo *uword* de *Armadillo*.
- **PSF:** variable del tipo *Cube* de *Armadillo*.
- **Sampling_grid:** variable del tipo *Cube* de *Armadillo*.

Al principio, se inicializa el cubo Sampling_grid como se observa en el siguiente código:

```
size_t n = qc.n_rows;
for(size_t i = 0; i < n ; i++){
    Sampling_grid(qc(i,0), qc(i,1), qc(i,2))= 1;
}
```

Lista 5.5: Inicializado de *Sampling_grid*.

Utilizando la matriz qc se va rellenando el cubo Sampling_grid. Cuando esta operación esta terminada se pasa a calcular el cubo PSF. Para ello, se harán una serie de llamadas sobre la transformada rápida de Fourier (FFT) y a su inversa (iFFT).

Los elementos menores que 0 en el cubo PSF (del inglés, Point Spread Function) se convierten en 0. Se realiza una multiplicación elemento a elemento entre los cubos PSF y Sampling_grid y finalmente se devuelve el valor PSF dividido por el acumulado de PSF, es decir, la suma total de todos sus elementos.

5.4.1.1. Construct SH basis

Esta función construye la matriz de base de los armónicos esféricos en puntos especificados. Recibe como parámetros:

- **degree:** variable del tipo *uword* de *Armadillo*. Indica el grado máximo de los armónicos esféricos.
- **sphere_points:** variable del tipo *Mat* de *Armadillo*. Indica el número de muestras en la esfera.
- **dl:** variable del tipo *short*. Indica si se realiza a banda completa (dl = 1) o solamente sobre los pares (dl = 2).

- **real_or_complex:** variable del tipo *short*. Indica si se están utilizando números reales o complejos, significando el 0 que son reales.
- **basisV:** variable del tipo *Mat* de *Armadillo*. Variable pasada por referencia que es donde se devolverá el resultado.

Se devuelve como resultado la matriz basisV, variable que es pasada por referencia.

5.4.1.2. Recon Matrix

Función para calcular la matriz utilizada en la inversión de armónicos esféricos. Recibe como parámetros:

- **Y:** variable del tipo *uword* de *Armadillo*.
- **L:** variable del tipo *Mat* de *Armadillo*.
- **Lambda:** variable del tipo T, es decir, puede ser *float* o *double*, dependiendo de la precisión elegida.

Devuelve un objeto del tipo *Mat* de *Armadillo* que corresponde con la matriz resultante de la inversión.

5.5. Common

Al contrario que los apartados anteriores, *common* es una clase que engloba varias funciones. Entre ellas se encuentran la función Cart2Sph que permite convertir unas coordenadas cartesianas a polares y un conjunto de funciones relacionadas con la transformada rápida de Fourier.

5.5.1. Cart2Sph

Esta función convierte las coordenadas en formato cartesiano a formato esférico. Recibe como parámetros:

- **x:** variable del tipo T, es decir, puede ser *float* o *double*, dependiendo de la precisión elegida. Corresponde con la coordenada x en formato cartesiano.

- **y:** variable del tipo T, es decir, puede ser *float* o *double*, dependiendo de la precisión elegida. Corresponde con la coordenada y en formato cartesiano.
- **z:** variable del tipo T, es decir, puede ser *float* o *double*, dependiendo de la precisión elegida. Corresponde con la coordenada z en formato cartesiano.
- **phi:** variable del tipo T, es decir, puede ser *float* o *double*, dependiendo de la precisión elegida.
- **theta:** variable del tipo T, es decir, puede ser *float* o *double*, dependiendo de la precisión elegida.

El resultado de las operaciones se devuelve en las variables *phi* y *theta*.

5.5.2. fft3D

Esta función realiza la transformada rápida de Fourier sobre un objeto 3D. Recibe como parámetros:

- **Signal:** variable del tipo Cube de Armadillo, que corresponde con el objeto 3D al que se le quiere aplicar la FFT.

Devuelve como resultado: un objeto 3D que corresponde con el resultado de realizar FFT sobre la variable *Signal* recibida.

5.5.3. fftshift3D

Reordena un objeto 3D en orden creciente de sus frecuencias. Recibe como parámetros:

- **x:** variable del tipo *Cube* de Armadillo. Corresponde con el objeto 3D que quiere ser ordenado.

Devuelve como resultado: un objeto 3D que corresponde con el resultado de realizar la reordenación de frecuencias sobre la variable x recibida.

5.5.4. ifftshift3D

Reordena de forma inversa un objeto 3D en orden creciente de sus frecuencias. Recibe como parámetros:

- **x:** variable del tipo *Cube* de Armadillo. Corresponde con el objeto 3D que quiere ser ordenado.

Devuelve como resultado: un objeto 3D que corresponde con el resultado de realizar la reordenación inversa de frecuencias sobre la variable x recibida.

5.5.5. ifftshift2D

Reordena de forma inversa un objeto 3D en orden creciente de sus frecuencias. Recibe como parámetros:

- **x:** variable del tipo *Mat* de Armadillo. Corresponde con el objeto 2D que quiere ser ordenado.

Devuelve como resultado: un objeto 2D que corresponde con el resultado de realizar la reordenación inversa de frecuencias sobre la variable x recibida.

5.5.6. fftshift2D

Reordena un objeto 2D en orden creciente de sus frecuencias. Recibe como parámetros:

- **x:** variable del tipo *Mat* de Armadillo. Corresponde con el objeto 3D que quiere ser ordenado.

Devuelve como resultado: un objeto 2D que corresponde con el resultado de realizar la reordenación de frecuencias sobre la variable x recibida.

5.5.7. fft2D

Esta función realiza la transformada rápida de Fourier sobre un objeto 2D. Recibe como parámetros:

- **Signal:** variable del tipo *Mat* de *Armadillo*, que corresponde con el objeto 2D al que se le quiere aplicar la FFT.

Devuelve como resultado: un objeto 2D que corresponde con el resultado de realizar FFT sobre la variable *Signal* recibida.

Capítulo 6

Pruebas

En este capítulo se va a evaluar el alcance del sistema una vez realizado el diseño (Capítulo 4) y la implementación (Capítulo 5). El resultado de las pruebas confirmará que todas las funcionalidades el sistema final se cumplen. Por último se muestra una matriz de trazabilidad entre pruebas y requisitos funcionales, que demuestra que las pruebas cubren todos los requisitos funcionales.

6.1. Entorno de pruebas

Las pruebas han sido realizadas en los siguientes equipos:

Tabla 6.1: Especificaciones equipo 1 de pruebas.

Nombre del equipo	Nodo de cómputo 9-1
Procesador	Intel Xeon CPU E5-2630 2.4GHz
GPU	GeForge GTX TITAN Black
Memoria principal	6 GigaBytes
Sistema operativo	Ubuntu 14.04

Tabla 6.2: Especificaciones equipo 2 de pruebas.

Nombre del equipo	Máquina Virtual Ubuntu
Procesador	2,6 GHz Intel Core i7
GPU	-
Memoria principal	2 GB
Sistema operativo	Ubuntu 14.04

6.1.1. MRIcron

MRIcron es una aplicación que nos permite visualizar imágenes en formato NIfTI, cómo las generadas con nuestra aplicación. Esta aplicación nos servirá para comprobar que las imágenes generadas por la aplicación resultan ser totalmente iguales a las generadas por la aplicación anterior.

En la Figura 6-1 se puede ver el programa mricron en ejecución. En él podemos movernos en las 3 direcciones del espacio para ver el resultado de la ejecución. Cómo se puede comprobar, MRICron nos muestra tres vistas del cerebro:

- **Vista coronal:** imagen superior izquierda.
- **Vista sagital:** imagen superior derecha.
- **Vista transversal:** imagen inferior.

6.2. Definición del plan de pruebas

En esta sección se va a presentar el formato en el que serán definidas las pruebas, así como las tablas de las pruebas realizadas.

Cada una de las pruebas cuenta con las siguientes características, tal y cómo muestra la tabla de ejemplo 6.3:

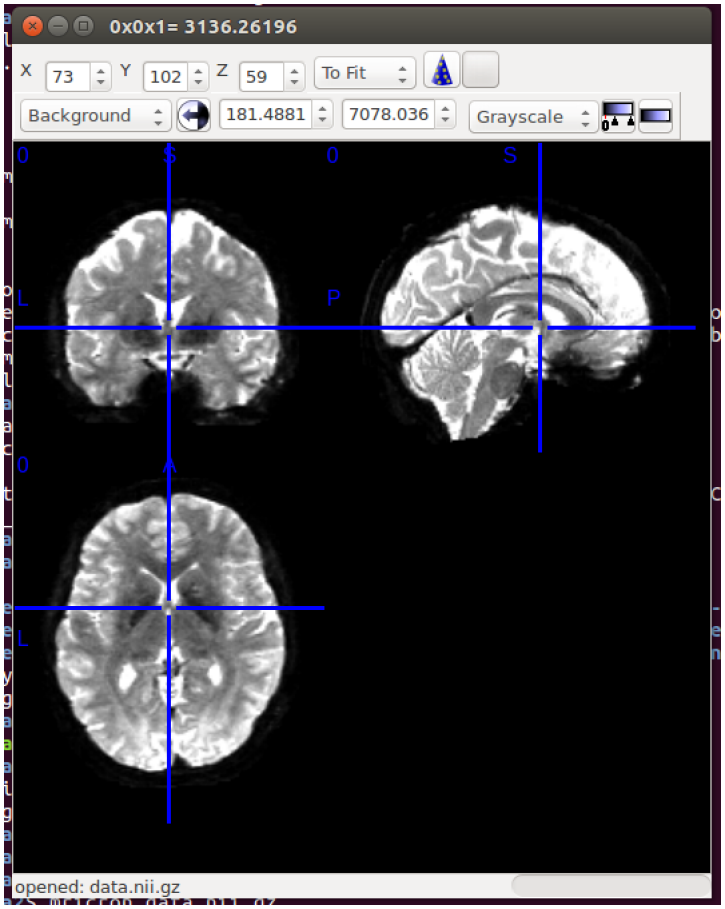

Figura 6-1: MRIcron. Aplicación para visionado de imágenes.

Tabla 6.3: Tabla ejemplo de especificación de las pruebas

Identificación	
Nombre	
Objetivo	
Requisitos relacionados	
Descripción	
Éxito:	

- **Identificación:** Código unívoco que sirve para identificar cada una de las pruebas. El formato de éste código es el siguiente:
  - **PR:** Las primeras dos letras indican que se trata de una prueba.
  - **XX:** las dos X siguientes representan la numeración de la prueba en referencia con el número total de pruebas.
- **Nombre:** Título a modo de breve resumen de la prueba.
- **Objetivo:** Finalidad principal de la prueba.
- **Requisitos relacionados:** Requisitos que están directamente o indirectamente relacionados con la prueba.
- **Descripción:** Explicación detallada de como se va a llevar a cabo la prueba y como se va a comprobar su éxito o fracaso.
- **Éxito:** Indica si la prueba ha sido realizada con éxito o no.

Tabla 6.4: Tabla prueba PR-01.

Identificación	PR-01
Nombre	Ejecutar *Rumba*
Objetivo	Comprobar que la aplicación permite la ejecución del programa utilizando el algoritmo *Rumba*.
Requisitos relacionados	RF-01
Descripción	1. Se lanzará una ejecución de la aplicación especificando en la opción *-a* que se quiere utilizar el algoritmo *rumba*. 2. Una vez ejecutada la aplicación se ira al fichero de logs a comprobar que el algoritmo que se utilizó fue *rumba*
Éxito:	Sí

Tabla 6.5: Tabla prueba PR-02.

Identificación	PR-02
Nombre	Ejecutar *dsi*
Objetivo	Comprobar que la aplicación permite la ejecución del programa utilizando el algoritmo *DSI*
Requisitos relacionados	RF-02
Descripción	1. Se lanzará una ejecución de la aplicación especificando en la opción *-a* que se quiere utilizar el algoritmo *dsi*. 2. Una vez ejecutada la aplicación se ira al fichero de logs a comprobar que el algoritmo que se utilizó fue *dsi*.
Éxito:	Sí

Tabla 6.6: Tabla prueba PR-03.

Identificación	PR-03
Nombre	Lectura de imágenes NIFTI
Objetivo	Comprobar que la aplicación permite la lectura de las imágenes en formato NIFTI
Requisitos relacionados	RF-03
Descripción	1. Se lanzarán ejecuciones en las cuales los datos de los cerebros estarán en ficheros con el formato NIFTI. 2. Las ejecuciones deben completarse con éxito. 3. Comprobar con el programa MRICron que las imágenes resultantes son correctas.
Éxito:	Sí

Tabla 6.7: Tabla prueba PR-04.

Identificación	PR-04
Nombre	Instalación en los sistemas Linux y Windows
Objetivo	Comprobar que la aplicación puede ser instalada en sistemas Linux y Windows
Requisitos relacionados	RF-04 y RF-05
Descripción	1. Instalar el sistema de la aplicación en sistemas Linux y Windows. 2. Ejecutar la aplicación. 3. Ver que los resultado de la ejecución son correctos.
Éxito:	Sí

Tabla 6.8: Tabla prueba PR-05.

Identificación	PR-05
Nombre	Ejecución sobre GPUs
Objetivo	Comprobar que la aplicación puede ser ejecutada sobre una GPU.
Requisitos relacionados	RF-06
Descripción	1. Ejecutar la aplicación con la opción de NVBlas. 2. Comprobar con un programa como nvidia-smi si se esta ejecutando la aplicación sobre la GPU.
Éxito:	Sí

Tabla 6.9: Tabla prueba PR-06.

Identificación	PR-06
Nombre	Elegir el método de lectura de datos
Objetivo	Comprobar que la aplicación permite la elección del método de lectura de datos.
Requisitos relacionados	RF-07, RF-08 y RF-09
Descripción	1. Lanzar la aplicación alternando con los distintos métodos de lectura posible (*slice*,*volume*,*voxel*) 2. Comprobar que en los Logs de la aplicación se especifica el método elegido.
Éxito:	Sí

Tabla 6.10: Tabla prueba PR-07.

Identificación	PR-07
Nombre	Elegir precisión de los datos
Objetivo	Comprobar que la aplicación permite la elección de la precisión de los datos.
Requisitos relacionados	RF-10, RF-11
Descripción	1. Lanzar la aplicación alternando con las distintas precisiones de datos posibles (*float*,*double*) 2. Comprobar que en los Logs de la aplicación se especifica el la precisión elegida.
Éxito:	Sí

Tabla 6.11: Tabla prueba PR-08.

Identificación	PR-08
Nombre	Ayuda de la aplicación
Objetivo	Comprobar que la aplicación muestra la ayuda cuando se introduce el parámetro ayuda.
Requisitos relacionados	RF-12
Descripción	1. Lanzar la ejecución indicando el parámetro de ayuda 2. Comprobar que se muestra por la terminal de comandos la forma de ejecución y los parámetros necesarios
Éxito:	Sí

Tabla 6.12: Tabla prueba PR-09.

Identificación	PR-09
Nombre	Compresión de los datos
Objetivo	Comprobar que la aplicación permite la compresión de los datos obtenidos.
Requisitos relacionados	RF-13
Descripción	1. Lanzar la ejecución indicando el parámetro de compresión 2. Comprobar en los logs de la aplicación que se ha elegido que los datos sean comprimidos al finalizar. 3. Comprobar que los datos comprimidos son correctos.
Éxito:	Sí

Tabla 6.13: Tabla prueba PR-10.

Identificación	PR-10
Nombre	Mejora de tiempos
Objetivo	Comprobar que la aplicación mejora los tiempos de ejecución respecto a la aplicación anterior.
Requisitos relacionados	RF-14
Descripción	1. Se lanzarán varias ejecuciones de la aplicación generada. 2. Se lanzarán varias ejecuciones de la aplicación anterior, escrita en *Matlab*. 3. Una vez terminadas las ejecuciones se comprobará que los tiempos de ejecución mejoran con la nueva aplicación.
Éxito:	Sí

6.3. Matriz trazabilidad pruebas-requisitos

La Tabla 6.14 muestra una relación entre las pruebas diseñadas y los requisitos funcionales, donde puede observarse que se comprueba el correcto funcionamiento de la aplicación.

Tabla 6.14: Matriz trazabilidad casos de uso requisitos.

	RF-01	RF-02	RF-03	RF-04	RF-05	RF-06	RF-07	RF-08	RF-09	RF-10	RF-11	RF-12	RF-13	RF-14
PR-01	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
PR-02	-	✓	-	-	-	-	-	-	-	-	-	-	-	-
PR-03	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
PR-04	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-
PR-05	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
PR-06	-	-	-	-	-	-	✓	✓	✓	-	-	-	-	-
PR-07	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-
PR-08	-	-	-	-	-	-	-	-	-	-	-	✓	-	-
PR-09	-	-	-	-	-	-	-	-	-	-	-	-	✓	-
PR-10	-	-	-	-	-	-	-	-	-	-	-	-	-	✓

Capítulo 7

Evaluación y rendimiento

El principal objetivo de este capítulo es el de mostrar la evaluación que se ha realizado de la aplicación y el rendimiento en comparación con la aplicación inicial. Estas evaluaciones se han realizado en el *clúster* del grupo de investigación ARCOS de la Universidad Carlos III de Madrid.

7.1. Evaluación de la aplicación

Para las pruebas de evaluación se contaba con dos datos tamaño distinto. Unos datos que llamaremos pequeños que ocupan 117 MegaBytes y unos datos que llamaremos grandes que ocupan 4 GigaBytes.

En las siguientes tablas se muestran las evaluaciones realizadas sobre la aplicación y los resultados obtenidos. Las evaluaciones han sido llevadas a cabo sobre el siguiente equipo:

Tabla 7.1: Especificaciones equipo de evaluación

Nombre del equipo	Nodo de cómputo 9-1
Procesador	Intel Xeon CPU E5-2630 2.4GHz
GPU	GeForge GTX TITAN Black
Memoria principal	6 Giga Bytes

En la Figura 7-1 podemos observar la evaluación de la aplicación utilizando los datos pequeños sobre el equipo GTX TITAN BLACK. En el se puede observar una comparación del tamaño de bloque de los datos de la GPU en comparación con el tiempo de ejecución. El mejor resultado se obtiene eligiendo el tamaño de *tile* de 1.024.

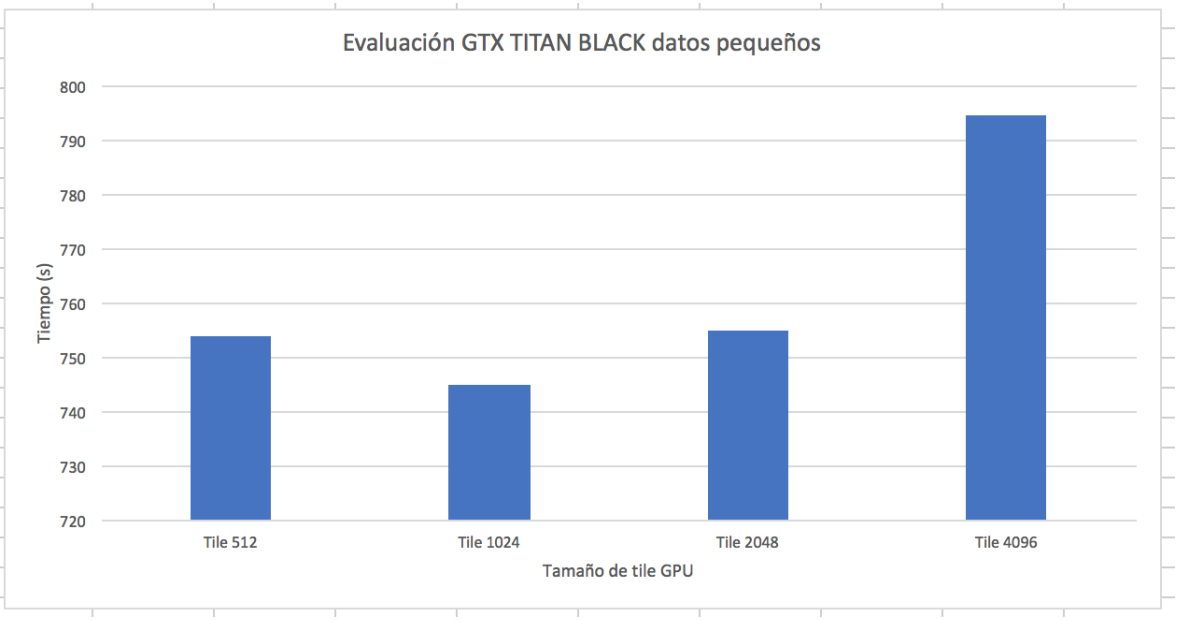

Figura 7-1: Comparación ejecuciones GPU de datos pequeños.

En la Figura 7-2, se muestra la comparación de las ejecuciones de los datos pequeños con la ejecución con el código en *Matlab*. La escala vertical es logarítmica debido a la descompensación entre los tiempos de ejecución sobre GPU y *Matlab*.

De igual modo, en la Figura 7-3 podemos observar que se obtienen los mejores resultado para un tamaño de *tile* de 1.024.

La Figura 7-4 muestra un gráfico, para comparar las ejecuciones con los datos grandes, en escala logarítmica. Esto es así porque el tiempo de ejecución de la aplicación en *Matlab*, correspondía con 48 horas.

7.2. Rendimiento de la aplicación

En las tablas siguientes se muestra el *speedup* obtenido resultado de la comparación entre las ejecuciones de la nueva aplicación sobre GPUs y los resultados obtenidos de las ejecuciones de la aplicación escrita en *Matlab*.

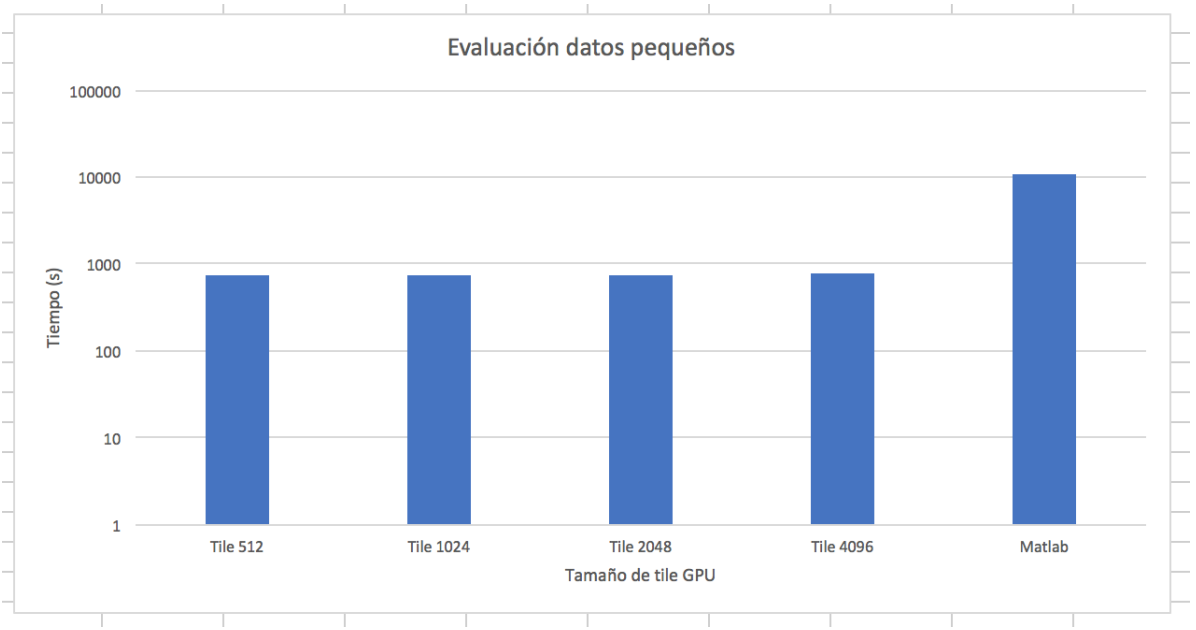

Figura 7-2: Evaluación datos pequeños.

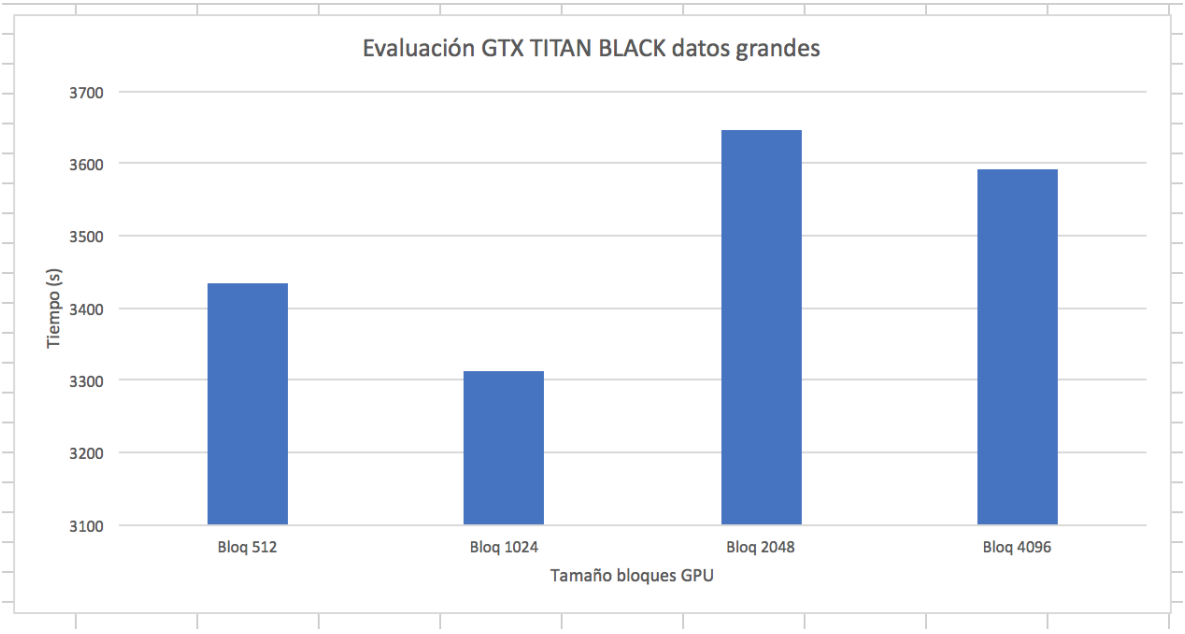

Figura 7-3: Comparación ejecuciones GPU de datos grandes.

7.2.1. Datos pequeños

El mejor resultado de los tiempos de ejecución con los datos pequeños corresponde al valor 744,99 segundos obtenido con un tamaño de *tile* de 1.024. La ejecución con la aplicación anterior tenía una duración de 3 horas, 10.800 segundos. De modo que el *speedup* conseguido respecto a los datos pequeños

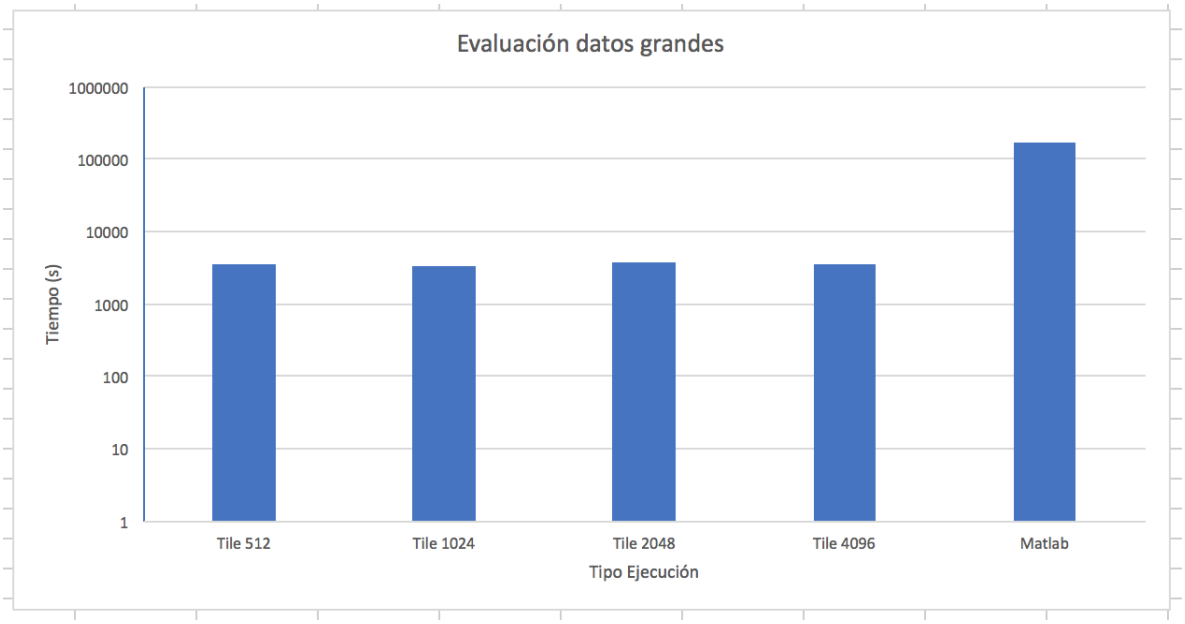

Figura 7-4: Evaluación datos grandes.

se puede observar en la Ecuación 7.1.

$$Speedup = \frac{10.800,00 segundos}{744,99 segundos} = 14,49 \tag{7.1}$$

7.2.2. Datos grandes

El mejor resultado de los tiempos de ejecución con los datos grandes corresponde al valor 3.312,58 segundos obtenido con un tamaño de *tile* de 1.024. La ejecución con la aplicación anterior tenía una duración de 48 horas, 172.800 segundos. De modo que el *speedup* conseguido respecto a los datos pequeños se puede observar en la Ecuación 7.2.

$$Speedup = \frac{172.800,00\ segundos}{3.312,58\ segundos} = 52,16 \tag{7.2}$$

Capítulo 8

Planificación y presupuesto del proyecto

En este capítulo se va a mostrar la planificación temporal que se ha llevada a cabo en la realización del proyecto. Se incluye un diagrama de GANTT de forma que pueda verse gráficamente el proceso de desarrollo del proyecto, que posteriormente es explicado mediante una tabla. Como sección final de este capítulo se muestra el presupuesto asociado al proyecto. Este presupuesto esta desglosado en la parte referente al personal implicado y en todo el coste de material.

8.1. Planificación

Este apartado se centrará en explicar la planificación temporal seguida para la realización del proyecto. La realización del proyecto empieza el día 1 de febrero de 2017 y se termina el día 20 de junio de 2017. De este modo, tenemos que la duración de la realización del proyecto es de 4 meses y 20 días. La dedicación diaria al proyecto era de 4 horas, sin tener en cuenta los días festivos y los fines de semana. El tutor del Trabajo de Fin de Grado dedicó una hora cada día a realizar el seguimiento del proyecto.

Para realizar de forma correcta y ordenada el proyecto lo primero es realizar una buena planificación del mismo. Esta tarea se llevó a cabo en 7 días. La forma en que se realizó el proyecto es una técnica denominada en cascada, lo que indica que las distintas etapas del proyecto van llevándose a cabo de forma ordenada y que no se pasa a la siguiente etapa sin haber terminado la anterior.

Una vez terminada la planificación estimada del proyecto, se realizó el diagrama de GANTT representado por la Figura 8-1, donde se puede ver de forma gráfica el seguimiento de cada tarea a lo largo del tiempo:

El diagrama de GANTT expresado en la Figura 8-1, muestra las tareas en el mismo orden en el que se encuentran en el presente documento.

Para una mejor comprensión de las etapas del proyecto se muestra la Tabla 8.1. En ella aparece como recuento total de días 130, sin embargo, como ya se ha especificado anteriormente hay que restarle los fines de semana y los días festivos, de modo que el total real de días es de 94 días.

Tabla 8.1: Fases del proyecto.

Fase	Actividad	Comienzo	Fin	Duración
Planificación		01/02/2017	07/02/2017	7 días
Estado del arte		08/02/2017	21/02/2017	14 días
Análisis del sistema		22/02/2017	14/03/2017	21 días
	Casos de uso	22/02/2017	28/02/2017	7 días
	Requisitos	01/03/2017	14/03/2017	14 días
Diseño		15/03/2017	20/03/2017	6 días
Implementación		21/03/2017	05/04/2017	16 días
Pruebas		06/04/2017	24/04/2017	19 días
	Diseño	06/04/2017	11/04/2017	6 días
	Satisfacción	12/04/2017	24/04/2017	13 días
Evaluación y rendimiento		25/04/2017	12/05/2017	18 días
	Evaluación	25/04/2017	09/05/2017	15 días
	Rendimiento	27/04/2017	12/05/2017	16 días
Presupuesto		15/05/2017	23/05/2017	9 días
	Costes de material	15/05/2017	19/05/2017	5 días
	Costes de personal	22/05/2017	23/05/2017	2 días
Impacto socio-económico		24/05/2017	26/05/2017	3 días
Marco regulador		29/05/2017	02/06/2017	5 días
Documentación		05/06/2017	20/06/2017	12 días
Total				130 días
Fines de semana y festivos				36 días
Total				**94 días**

- **Planificación:** etapa encargada de la planificación total del proyecto. Se dedicaron 7 días a esta tarea.

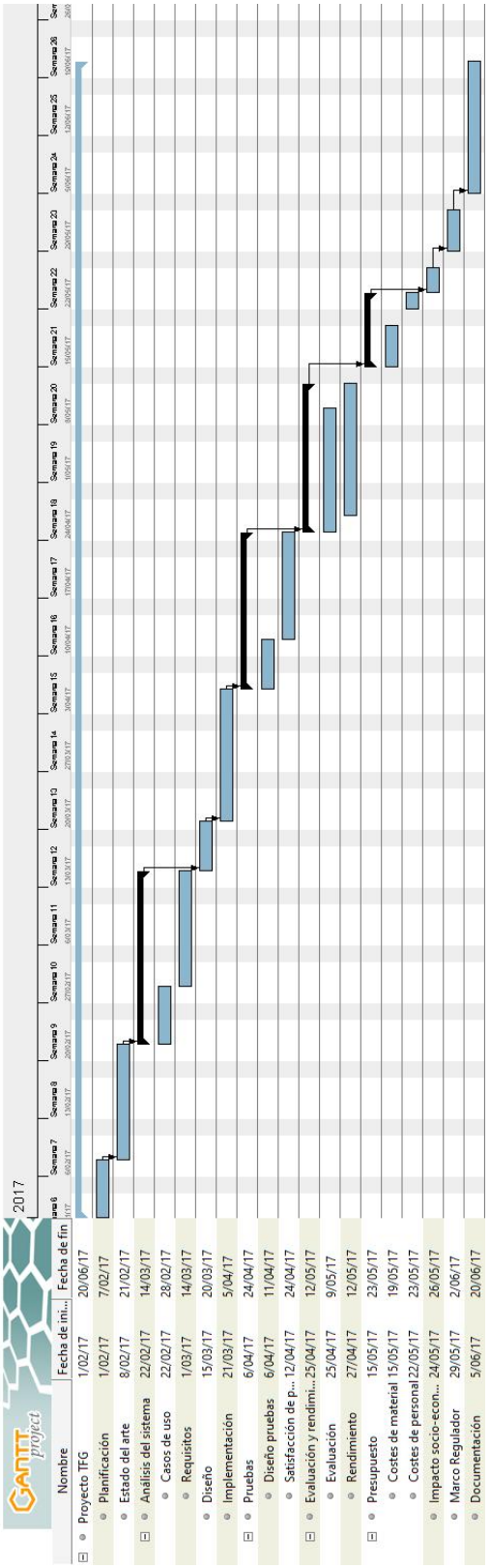

Figura 8-1: Diagrama de GANTT.

- **Estado del Arte:** es una de las etapas importantes. En ella, se realiza un estudio sobre las tecnologías relacionadas que pueden verse involucradas en el proyecto. Se dedicaron 14 días.
- **Análisis del sistema:** en ella se realizaron las reuniones con el tutor y se plasmaron las necesidades del proyecto en requisitos y casos de uso. Se dedicaron 21 días para esta tarea.
- **Diseño:** una vez definidos los requisitos, es necesario llevar a cabo un diseño del sistema en el cual se indiquen cuales son las clases involucradas y las relaciones entre ellas. Se dedicaron 6 días para esta etapa.
- **Implementación:** terminado el diseño es función del programador plasmar las ideas sobre código. Se dedicaron 16 días para esta tarea.
- **Pruebas:** etapa del proyecto que está destinada a diseñar y ejecutar las pruebas. Se dedicaron 19 días para esta etapa.
- **Evaluación y rendimiento:** etapa dedicada a comprobar el rendimiento y mejora que presenta el la aplicación implementada en referencia con la aplicación original. Se destinaron 18 días para esta tarea.
- **Presupuesto:** estudio sobre los costes de personal y los costes materiales asociados al proyecto. Se llevó a cabo en 9 días.
- **Impacto socio-económico:** esta etapa sirve para explicar el impacto que el proyecto tiene sobre el ámbito social. Se utilizaron 3 días.
- **Marco Regulador:** estudio de todos los aspectos legales que involucran al proyecto. Para esta etapa se destinaron 5 días.
- **Documentación:** etapa destinada a la realización de la memoria y los apartados de introducción, motivación, conclusiones,... que no son mencionados en la Tabla 8.1.

8.2. Presupuesto del proyecto

En este apartado del capítulo se van a explicar de forma detallada los costes asociados a la realización del proyecto.

8.2.1. Costes desglosados

Pasamos ahora a desglosar el coste total del proyecto en dos categorías principales como son costes de material y costes de personal.

8.2.1.1. Costes de material

Los costes de material hacen referencia a aquellos gastos para obtener software, hardware o cualquier tipo de material fungible.

La Tabla 8.2 muestra los costes de material fungible utilizado durante el proyecto.

Tabla 8.2: Costes material fungible.

Material	Unidades	Coste unitario	Coste
Boligrafo Pilot gel	2	1,20 €	2,40 €
Cuaderno A4 Oxford	1	4,36 €	4,36 €
Subrayador	2	0,75 €	1,50 €
Total			**8,26€**

La Tabla 8.3 muestra todos los costes asociados a los equipos informáticos utilizados durante el proyecto. Se ha estimado que el material informático será amortizado a 5 años.

Tabla 8.3: Costes material informático.

Material	Precio	Amortización 1 año	Meses utilizado	Coste
Macbook Pro	1.500,00 €	300,00 €	4	100,00 €
Ordenador sobremesa	870,00 €	174,00 €	4	58,00 €
Geforce GTX Titan	918 €	183.6 €	4	61,2 €
Total				**219,20€**

8.2.1.2. Costes de personal

Estos costes hacen referencia a todos los gastos asociados a contratos de personas para la realización del proyecto. Como se ve en la tabla 8.4, existen múltiples roles, que aunque algunos son realizados por la misma persona, es necesario desglosarlos por la diferencia de salario entre ellos. Estos roles son:

- **Administrador del proyecto:** rol que lleva a cabo el tutor del Trabajo Fin de Grado. Se encarga de supervisar todo el proyecto.

- **Analista:** su principal función es la de teniendo en cuenta las necesidades requeridas con el cliente generar los casos de usos y requisitos del sistema. El autor del proyecto realiza este rol.
- **Diseñador:** una vez expresadas las necesidades del sistema por parte del analista, el diseñador es el encargado de realizar los diseños de clases, diseño del sistema. El autor del proyecto realiza este rol.
- **Programador C++:** su función es la de plasmar en código los diseños generados por el diseñador. El autor del proyecto realiza este rol.
- **Tester:** una vez que todo el código está generado, es el encargado de probar que todo funciona como se especifica en los requisitos, casos de uso y que se cumplen todas las necesidades. El autor del proyecto realiza este rol.

Para la estimación de los salarios de cada rol se utilizó el documento "GUIA HAYS 2017 INFORME SECTORES Y SALARIOS", documento que se puede descargar en la página oficial [6].

Tabla 8.4: Costes asociados a personal.

Rol del proyecto	**Horas estimadas**	**Coste por hora (€/hora)**	**Total (€)**
Administrador del proyecto	94	33,85	3.181,90 €
Analista	228	16,34	3.725,52 €
Diseñador	16	19,95	319,20 €
Programador C++	48	12,50	600,00 €
Tester	108	13,46	1.453,68 €
Total	**494**		**9.280,30 €**

8.2.2. Coste total asociado al proyecto

El proyecto al que hace referencia el presente documento se llevó a cabo en un período de 4 meses y 20 días.

El coste final asociado al proyecto es de 15.611,44 €(**QUINCE MIL SEISCIENTOS ONCE EUROS CON CUARENTA Y CUATRO CÉNTIMOS DE EURO**).

Tabla 8.5: Coste total del proyecto.

Coste del proyecto	
Costes de personal	9.280,30 €
Costes de material fungible	8,26 €
Costes de material informático	219,20 €
Riesgo (18 %)	1.711,39 €
Beneficios (15 %)	1.682,87 €
Total sin IVA	12.902,02 €
IVA (21 %)	2.709,42 €
Total	15.611,44 €

Capítulo 9

Marco regulador e impacto socio-económico

Este capítulo del documento se centra en los aspectos legales referentes al proyecto. Se ha estructurado tal y como se expresa en el documento matriz de evaluación teniendo en cuenta la legislación aplicable sobre la implementación, explicando los estándares técnicos utilizados y comentando la propiedad intelectual del proyecto. En la sección final del capítulo se comentará el impacto social que tiene este proyecto.

9.1. Legislación aplicable sobre la implementación descrita

Dada la naturaleza de este proyecto, se necesitan imágenes de pacientes para realizar los cálculos de las ejecuciones. Las imágenes utilizadas y de las cuáles se muestran ejemplos en el presente documento han sido anonimizadas y no es posible llegar a la fuente que cedió tales imágenes.

Por tratarse de datos médicos, éstos están regulados bajo la Ley Orgánica de Protección de Datos [2].

9.2. Licencias y tecnologías empleadas

Este apartado hace referencia a las licencias de las tecnologías utilizadas para llevar a cabo la implementación del proyecto. Las tecnologías utilizadas bajo licencia son las siguientes:

- **Microsoft Windows:** Sistema operativo. Para el desarrollo del sistema se utilizó una licencia de estudiante cedida por la Universidad Carlos III de Madrid.
- **Paquete Office de Windows:** conjunto de herramientas de ofimática utilizado para generar algunas tablas y gráficos. Se utilió una licencia de estudiante cedida por la Universidad Carlos III de Madrid.
- **Matlab:** Entorno de programación para el lenguaje Matlab. Para el desarrollo del sistema se utilizó una licencia de estudiante cedida por la Universidad Carlos III de Madrid.
- **Notepad++:** Programa de edición de texto, utilizado para la realización del código. Presenta una licencia libre que se muestra en el acuerdo de licencia, al instalar la aplicación.
- **MRIcron:** programa que permite visualizar imágenes NIfTI. Usado en el proyecto para comprobar que las imágenes generadas eran correctas.
- **CMake:** programa utilizado para la generación de código de compilación. Cuenta con una licencia BSD, es decir,
- **ShareLaTeX:** utilizado para la realización de la memoria. Se utilizó una licencia personal que es un licencia gratuita. Esta licencia permite la existencia de un colaborador, que corresponde con el tutor del proyecto.
- **C++:** es un lenguaje de licencia libre. Con el se ha llevado a cabo la implementación de la aplicación.
- **Visual Paradigm:** utilizado para realizar los diagramas de clases del proyecto. Se utilizó la licencia de prueba de 30 días.
- **Gantt project:** programa utilizado para la generación del diagrama de GANTT. Es un programa sin licencia.

9.3. Propiedad intelectual del proyecto

El presente proyecto cuenta con una licencia que puede observarse en la dirección web del proyecto [4].

9.4. Impacto social

El tutor del TFG, autor de *pHARDI* y los demás coautores mantienen la idea de que la aplicación sea de libre distribución, de forma que cualquier hospital o grupo de investigación pueda utilizar esta tecnología y beneficiarse de ella.

Realizar una imagen de resonancia magnética, conlleva que el paciente, debe permanecer inmóvil en el dispositivo durante, en media de 30 a 60 minutos. La reconstrucción posterior y procesado de la organización del cerebro es costosa en cómputo y por tanto en tiempo. Esto puede demorar, cómo ya se ha visto, dependiendo del tamaño de los datos de horas a días.

Gracias a *pHARDI* y la hipótesis de **utilizar GPUs para la aceleración del procesado de estas imágenes**, introducida como título de este proyecto se consigue reducir esos tiempos y por tanto, poder ofrecer un diagnóstico en un menor tiempo.

El código puede ser descargado desde el repositorio [5] del grupo ARCOS de la Universidad Carlos III de Madrid.

Capítulo 10

Conclusiones

En el capítulo final del documento se recogen las conclusiones del alumno tras realizar el proyecto y un estudio de los trabajos futuros que puedan mejorar la aplicación propuesta. Como conclusiones finales del trabajo podemos concretar que la hipótesis propuesta por el tutor es cierta, y tal y como se muestra en el capítulo de Evaluación y Rendimiento (Capítulo 7), utilizando datos pequeños, se obtiene una *speedup* del 14,49 y utilizando datos grandes se obtiene un *speedup* de 52,16.

Como ya se explicó en el capítulo de Marco regulador e Impacto Socio-económico (Capítulo 9), el código está abierto para que cualquier persona pueda aportar nuevo conocimiento y para que cualquier hospital o centro de investigación pueda utilizar la aplicación.

10.1. Conclusiones personales

El principal objetivo del Trabajo Fin de Grado, tal y como se explica en el capítulo de introducción 1, es el de acelerar un procesador de imagen de resonancia magnética mediante tarjetas gráficas o GPUs, lo que se ha conseguido en este proyecto según las condiciones y resultados descritas en el Capítulo Evaluación y rendimiento 7.

La parte más importante del proyecto considero que ha sido el estudio del estado de arte y la adquisición de conocimiento de las tecnologías que se iban a utilizar durante el proyecto, lo que es fundamental y básico a la hora de realizar cualquier proyecto, ya que proporciona una base sólida de la cual parten las hipótesis del presente Trabajo de Fin de Grado. Otra parte importante del proyecto ha sido la de análisis del mismo,

para el cuál se utilizaron los conocimientos adquiridos en las asignaturas de Ingeniería del Software.

La realización del Trabajo de Fin de Grado, ha permitido al alumno afianzar conocimientos adquiridos durante la carrera y recordar algunos conocimientos que sólo pueden ser practicados con la realización de este tipo de proyectos. Además, el alumno ha tenido la oportunidad de utilizar tecnologías que aunque se explican como teoría durante la carrera no son objetivo de prácticas o trabajos. En el caso del presente proyecto puede aplicarse a la programación y utilización de GPUs.

Otras tecnologías utilizadas durante la realización ya eran conocidas por el alumno. Este es el caso de por ejemplo técnicas de paralelización de código mediante la utilización de *OpenMP*, el uso de herramientas de modelado de diagramas o extracción de requisitos.

10.2. Trabajos futuros

En esta sección del documento se enumeran y explican algunas propuestas destinadas a mejorar y/o complementar el proyecto de aceleración de un procesador de imagen por resonancia magnética. A continuación se muestran algunos de los posibles trabajos futuros a realizar:

- Una de las propuestas para trabajos futuros pretende eliminar algunas dependencias con paquetes que se utilizan para la ejecución de la aplicación. Este es el caso del paquete *ITK* el cuál es únicamente utilizado para la lectura de las imágenes en formato *NIFTI*. Para ello, se desea desarrollar una biblioteca que permita leer este formato de imágenes y así no depender de un paquete tan pesado como *ITK*.
- Utilizar FPGAs para la aceleración por hardware de las ejecuciones. Las FPGAs son un tipo de dispositivo hardware que puede ser programado para cumplir una determinada función. Un futuro Trabajo Fin de Grado podría basarse en la aplicación de FPGA para acelerar el procesador de imagen.
- Interfaz gráfica. Se desea desarrollar una interfaz gráfica que facilite la ejecución de la aplicación y el visionado de las imágenes generadas.
- Facilitar la instalación de la aplicación y el entorno necesario. De este modo, se conseguiría también que personas sin un alto conocimiento informático puedan beneficiarse de la aplicación.

Anexo 1: Summary in English

Abstract

This document recollects the Bachelor Thesis entitled “Aceleración de un procesador de Imagen de Resonancia Magnética mediante GPUs”.

It was first carried out a study of the state of the art on the technologies and methods that will be useful to demonstrate the proposed hypotheses. In this study we compared the main linear algebra libraries and technologies that allow the execution of code on GPUs. Secondly, a series of meetings were held with the thesis advisor to define the requirements. These requirements were embodied in the requirements description. Once the requirements were accepted by the advisor, it has been made the design and implementation of the solution. Next, the necessary tests were designed and carried out to verify that the system complied with the specifications agreed with the advisor.

In order to demonstrate the hypothesis defined by the title of the present project, a performance evaluation was done, in which the results of the application on GPU devices demonstrate the advantages of the implemented prototype. These results show that the hypothesis is true and that execution times are highly reduced compared to the previous version of the application.

This document also shows a schedule planning of the project, budget associated with it, a chapter dedicated to the legal aspects of the project, and a chapter dedicated to the socio-economic impact of the project.

Introduction

This chapter provides an overview of the project. Here it be will explained the motivation that leads to an accomplishment and the goal that to be achieved. It will also explain the structure that will follow the document and a list of the most commonly used terms and acronyms.

Motivation

Over the years, improvements in technology and advances in science have enabled the medical world to make diagnoses faster, safer, and less invasive for patients. Thanks to the use of new methods in modern medicine, we are able to generate images of the interior of the human body without the need of using an invasive technique for the patient.

This technique, called MRI (Magnetic Resonance Imaging) allows images of the soft tissues of the human body using, as its name suggests, a powerful magnet. The use of intravoxel reconstruction methods based on diffusion magnetic resonance data allows a study of the organization of the white matter of the brain. Further study of brain fiber connections allows the diagnosis of diseases such as schizophrenia or bipolar disorders.

Images taken may be considered as arrays of numbers. Conventional computer scan count with CPU (Central Processing Unit). CPUs perform sequential and limited parallel operations on one data at a time. However, there are other devices known as GPU (Graphics Processor Unit) that are specialized in performing the same operation on several data at a time. This technique known as SIMD (Single Instruction Multiple Data) proves to be beneficial for performing calculations on matrices. As prevously indicated, the images taken by the resonances can be considered arrays of numbers and the use of these devices can be really benefitial. In the particular case of this project, when using the GPUs to perform the calculations on the images taken by magnetic resonance, results in an improvement of the execution time and therefore of the calculation time.

On the other hand, *Matlab* has been chosen in many fields of science as standard language. Although it allows a rapid deployment of prototypes, *Matlab* due to its nature is a language that is strongly dependent on its own execution engine, which makes that although it is used on high performance equipment, it does not deliver the results Expected.

Therefore, the main motivation of this work is thanks to a high level language like C++ and the use of GPU accelerators, we can improve the execution times of a magnetic resonance image processor.

Objectives

The main objective of this work is **to analyze the possibility of acceleration of a Magnetic Resonance Image Processor with GPUs**. As secondary objectives within the above:

- Porting the existing application to a more optimized language.
- Use linear algebra libraries to compute and accelerate the calculation of matrices.

This document presents the design carried out to make the improvement on the code. Also, the images generated with the improvement offered must be totally the same as those generated with the previous method.

State of the art

In this chapter, it will be explained a significant number of technologies that allow to use C++ as a language with the final objective of performing calculations of linear algebra. In addition, several technologies will be explained that allow matrix calculations to be performed on GPUs rather than on CPUs.

Related work

The project's advisor has already done applications portability work in search of an improvement. In the paper [3], the advisor presents a series of solutions on an application written in *Matlab*. In this paper, he perform a translation of the algorithm RUMBA-SD, of a set of medical imaging tools called HARDI. In addition, the tutor presents performance evaluation with different solutions and finally it is verified that the version with Arrayfire is the one that offers better results due to the use of GPUs.

Intravoxel Fiver Reconstruction (RUMBA)

It is a method of reconstruction that allows to know the structural organization of the white matter of the brain. Knowing the connections between the nerve fibers of the brain helps diagnose mental illnesses such as schizophrenia and bipolar disorders.

As explained in the document [13] and in the research paper [3], in biological tissues, the diffusion of water molecules is not carried out in a similar way in all directions because of presence of natural barriers. This phenomenon can be explained quantitatively using the DTI diffusion tensor imaging. The representation of this tensor is a symmetric 3x3 positive matrix which can be seen in Figure 10-1.

$$D = \begin{bmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{bmatrix}$$

Figura 10-1: DTI matrix.

This diffusion tensor can be visualized as an ellipsoid, in which the orientations of the three main axes are defined by their eigenvectors and the length of the axes is defined by their eigenvalues.

The diffusion tensor model is adequate in case each brain, *voxel* has a single group of parallel fibers (See panel a of Figure 10-2). In this case, the main axis of the ellipsoid would correspond to the orientation of the fiber group. However, in regions where several fibers intersect, the diffusion tensor model does not allow calculating the orientation of these fibers (see panel b of Figure 10-2).

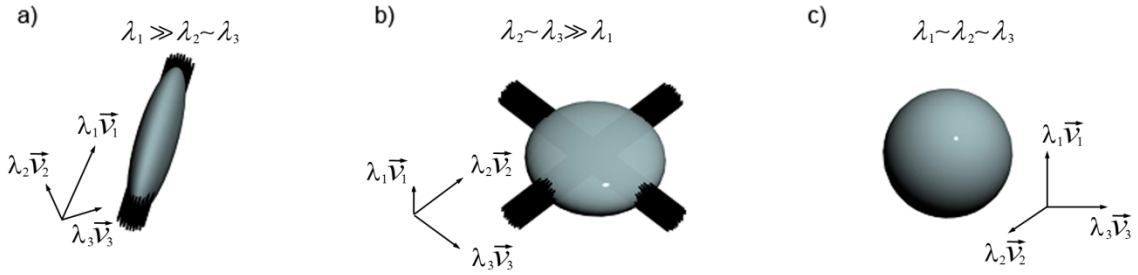

Figura 10-2: The microstructural organization can be revealed by the diffusion tensor ellipsoid.

Due to the limitations of DTI, the development of sampling protocols, diffusion models and reconstruction techniques has been promoted. One of these methods is RUMBA-SD whose objective is to obtain for each *voxel* a function of distribution of the orientation of the brain fibers.

As shown in Figure 10-3, to facilitate visualization, the color code of the surface is assigned according to the direction of diffusion ($[x, y, z] - [r, b, G]$, where r = red represents the probability along the left-to-right orientation, b = blue represents upper-lower orientation, and g = green represents orientation Antero posterior).

Figura 10-3: Orientation distribution function

C++ programming language

C++ is a high-level programming language. C++ is a very powerful language and has a very active support community so it is continuously adding new features and new optimizations. One of the possibilities offered by C++ that is really interesting for the completion of the Bachelor Thesis is the ability to make code using *templates*. As will be explained later, it is necessary that calculations can be performed with a precision that its decided by the user. It can be *float* or *double*. C++ *templates* are useful in that case. Another advantage of C++ is that it is a free license language.

Linear algebra libraries

Matlab

Matlab is a programming language and framework widely used in the scientific environment due to the wide range of tools it provides. *Matlab* has its own development environment, which enables the development and execution of scripting-like code. It has a large support and help community in Spanish, which can be useful in understanding the code.

Despite being a powerful tool for performing heavy calculations, *Matlab* is not fully optimized to perform critical tasks. This is due to *Matlab* has a high dependency on its own execution engine. Therefore, although applications written in the *Matlab* language were executed on high-performance platforms, resources would not be fully exploited because of their nature.

The main objective of this Bachelor Thesis is the acceleration of an MRI image processor and that is why it is decided as an initial measure to carry the code written in *Matlab* to a higher level and more optimized language as is C++.

```
a = [1 2 3; 4 5 6]

b = a(:,2) * 2
```

Lista 10.1: Basic operations on matrices.

In the List 10.1, we can observe basic operations with *Matlab*. The first operation consists of a definition of a two-dimensional matrix, the second operation shows a vector creation corresponding to the second column of a multiplied by 2.

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad b = \begin{bmatrix} 4 \\ 10 \end{bmatrix} \tag{10.1}$$

The result of operations 10.1 can be seen in the Equation 10.1.

From the main page of *Matlab* [8], we can download a code version. For the completion of the project, we have used a license offered by the Carlos III University of Madrid.

GNU Octave

GNU Octave is a programming language with a syntax very similar to Matlab. One advantage that GNU Octave offers against Matlab is that it is free software and therefore no license is required to use it. However, Octave is not so used in the scientific world because it does not offer the range of tools *Matlab* has.

Armadillo

Armadillo is a linear algebra library, written in C++, which provides multiple tools to perform calculations on both vectors and matrices. On of the main advantages of using the Armadillo library is that its semantics and many of the functions it provides have a great similarity to the functions in *Matlab*. This features facilitates the code porting and at the same time so that people who have experience with the *Matlab* syntax can fully understand the code written using the *Armadillo* library.

In the official *Armadillo* website [14], there is a section that contains a table of syntax equivalences

between *Matlab / Octave* and the syntax with *Armadillo*. *Armadillo* has its own data types such as *matrix*, *vector* or *cube*, which is very useful in the execution of the project since three-dimensional manipulation is required for the calculation.

In Listing 10.2, we can see an example code that is available on the official *Armadillo* web site. In this list we can see operations such as generation of random matrices of a chosen size (Mat A = randu textgreater (4,5)), multiplied by matrices (A * B), and the transposed operation of a matrix (B.t()).

```
#include <iostream>
#include <armadillo>

using namespace std;
using namespace arma;

int main()
{
  mat A = randu<mat>(4,5);
  mat B = randu<mat>(4,5);

  cout << A*B.t() << endl;

  return 0;
}
```

Lista 10.2: Example code *Armadillo*.

Parallel Computing Libraries and Languages

Arrayfire

Arrayfire is a high-performance library for parallel computing. Allows you to perform calculations using *CUDA* and *OpenCL* Kernels. For this, we have to configure the *backend*, which can be *CUDA*, *OpenCL* or CPU, in that order of preference.It is a pretty good solution since it has its own calls to create the Kernels needed to perform calculations with GPUs.

CUDA

The official *CUDA* [11] website defines this technology as a parallel computing architecture that benefits from GPUs. It belongs to the NVidia brand and can only be used on devices of the same brand. In order

to use this technology it is necessary to download the development drivers from the official website. *CUDA* can be used on Windows, Linux and MAC platforms. One of the advantages of using *CUDA* is that it can be implemented on high level languages such as C, C++ or *Fortran*, which implies that it is a technology that takes into account for our solution.

OpenCL

It is the standard parallel computing solution. Unlike *CUDA* it is possible to be used on any device. It presents an integration with C++ which makes it an interesting solution. Just as *CUDA* is available for Windows, Linux and MAC platforms.

Intel MKL

It is a library of *Intel* that aims to accelerate mathematical computing. Because Intel owns the software, the library is optimized to be used on systems with a processor of the same brand. For performing computing on CPU is very useful, however, the purpose of this project is to use GPUs for acceleration. Therefore, and despite being a successful solution, it is not the one sought.

From its official website [7], you can download the library for free.

BLAS

Basic Linear Algebra Subprogram. It is a library that has 3 levels of routines dedicated to perform basic operations on vectors and/or matrices.

- **Level 1:** which implements operations between scalar numbers, vectors and vector-vector operations.
- **Level 2:** Implements vector-matrix operations.
- **Level 3:** Implements matrix-matrix operations.

From the official website of the library [9], you can download the latest version of code.

NVBlas

NVBlas is a library that implements *BLAS* level 3 routines. This is because these routines present better potential for acceleration with GPUs. Allows integration with emph Armadillo. To accomplish this, it is necessary to add a configuration file in the same path of the application binary. It is also necessary to indicate when executing the program that loads the *NVBlas* dynamic library as follows:

```
LD_PRELOAD=/path\_to\_library\_NVBlas/libnvblas.so ./phardi
```

The system path of the library is dependent on the system on which it is run. In our case it corresponds to /usr/local/cuda/lib64/.

In the official website of *NVBlas* [12] we can find an extensive documentation that among other things tells us how to generate the configuration file for the use of this library in our applications.

LAPACK

Linear Algebra PACkage. It is written in FORTRAN 90 and has a large number of routines to solve systems of linear equations. As explained in the official LAPACK [10] website, the main purpose of this library is to make the EISPACK and LINPACK libraries run code efficiently on shared memory computers and parallel processor computers. EISPACK and LINPACK are inefficient in these computers because of the way they have access to the data in memory. The LAPACK routines are written in such a way that when a calculation is performed, it will be done by the Basic Linear Algebra Subprogram (BLAS) library. LAPACK is developed in a way that takes advantage of Level 3 of BLAS, level dedicated to operations matrix-matrix. It has a C ++ implementation called *lapack++*.

Evaluation and performance

The main objective of this chapter is to show the evaluation of the application and the performance compared to the initial application. These evaluations were carried out in the *cluster* of the research group ARCOS of the Carlos III University of Madrid.

Evaluation of the application

Two different size data were available for the evaluation tests. Some data that we will call small data that occupy 117 MegaBytes and some data that we will call large data that occupy 4 GigaBytes.

The following tables show the evaluations made on the application and the results obtained. The evaluations have been carried out on the following equipment:

Tabla 10.1: Specifications of the evaluation equipment.

Name of the equipment	Nodo de cómputo 9-1
Proccessor	Intel Xeon CPU E5-2630 2.4GHz
GPU	GeForge GTX TITAN Black
Main memory	6 Giga Bytes

In Figure 10-4 we can see the evaluation of the application using the small data on the GTX TITAN BLACK equipment. In it you can see a comparison of the block size of the GPU data compared to the runtime. The best result is obtained by choosing the Tile size of 1,024.

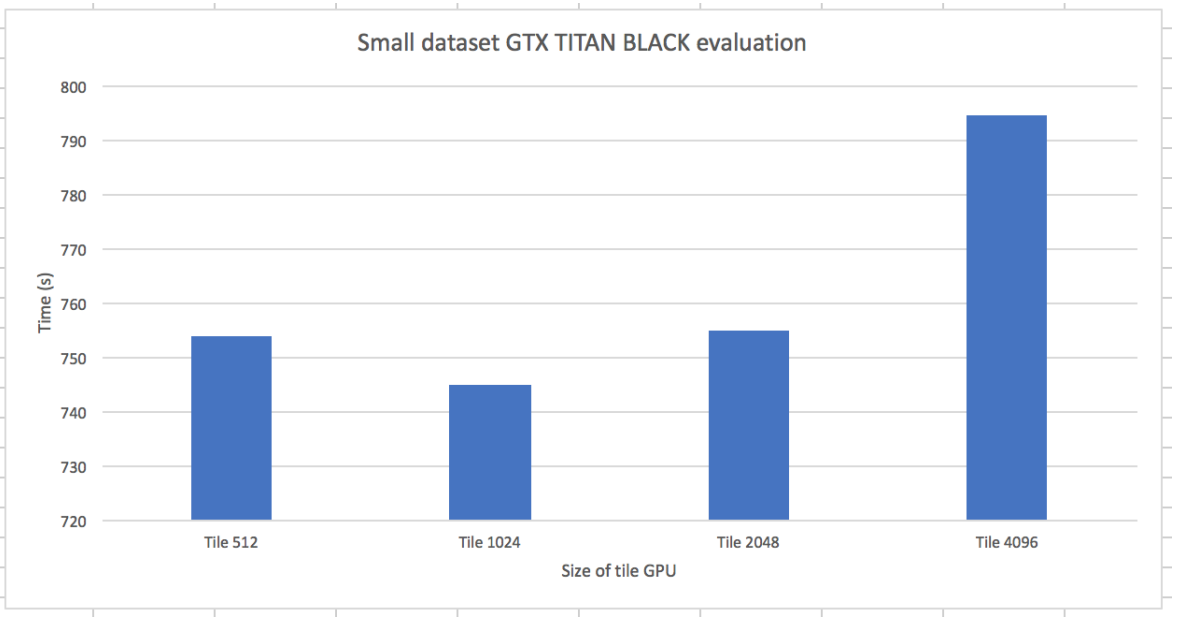

Figura 10-4: Comparison GPU execution small dataset.

In Figure 10-5, the comparison of small dataset execution with execution with the code in *Matlab* is

shown. The vertical scale is logarithmic due to the decompensation between execution times over GPU and *Matlab*.

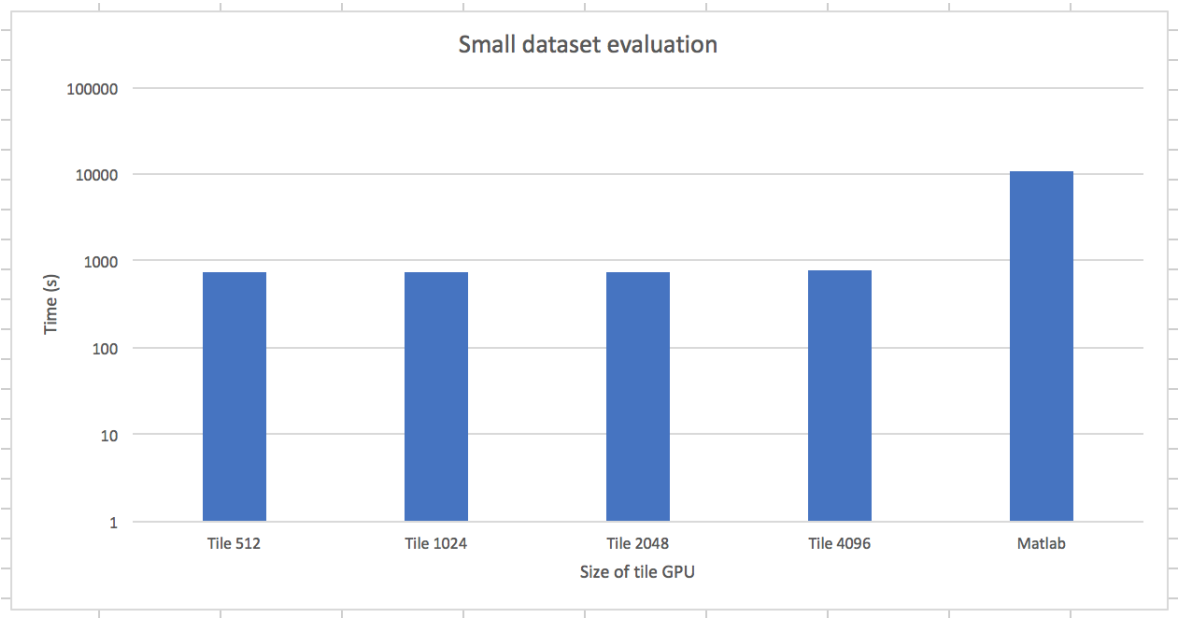

Figura 10-5: Small dataset evaluation.

Likewise, in Figure 10-6 we can see that the best result is obtained for a size of Tile of 1.024.

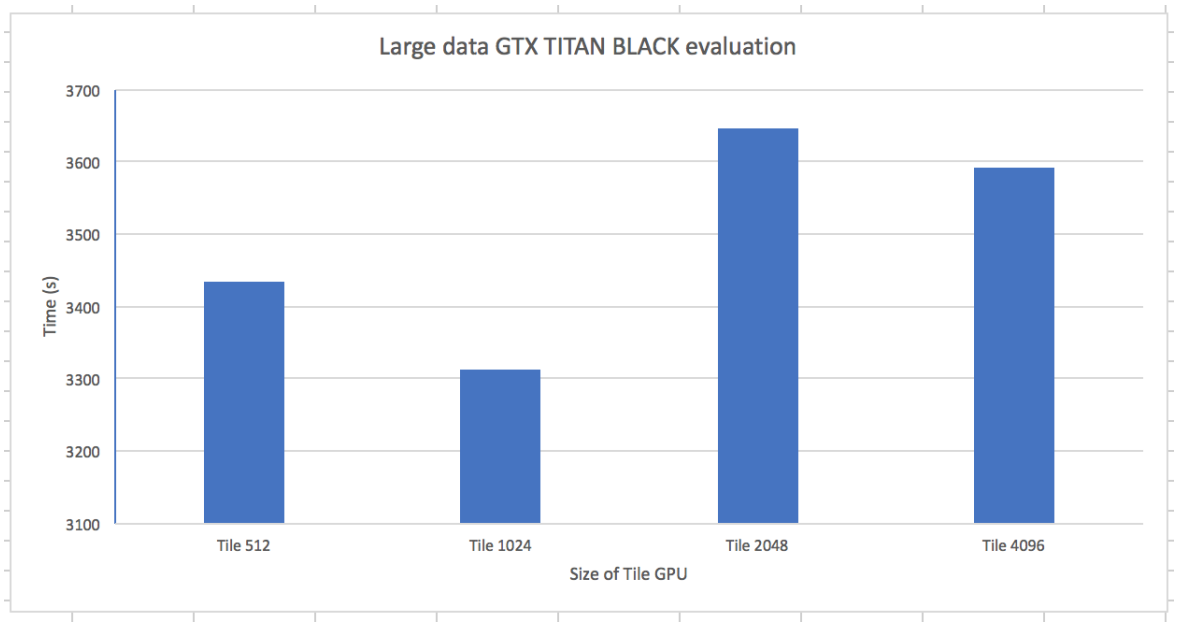

Figura 10-6: Comparison of large dataset GPU executions.

Figure 10-7 shows a graph, to compare runs with large data, in logarithmic scale. This is because the execution time of the application in *Matlab*, corresponded to 48 hours.

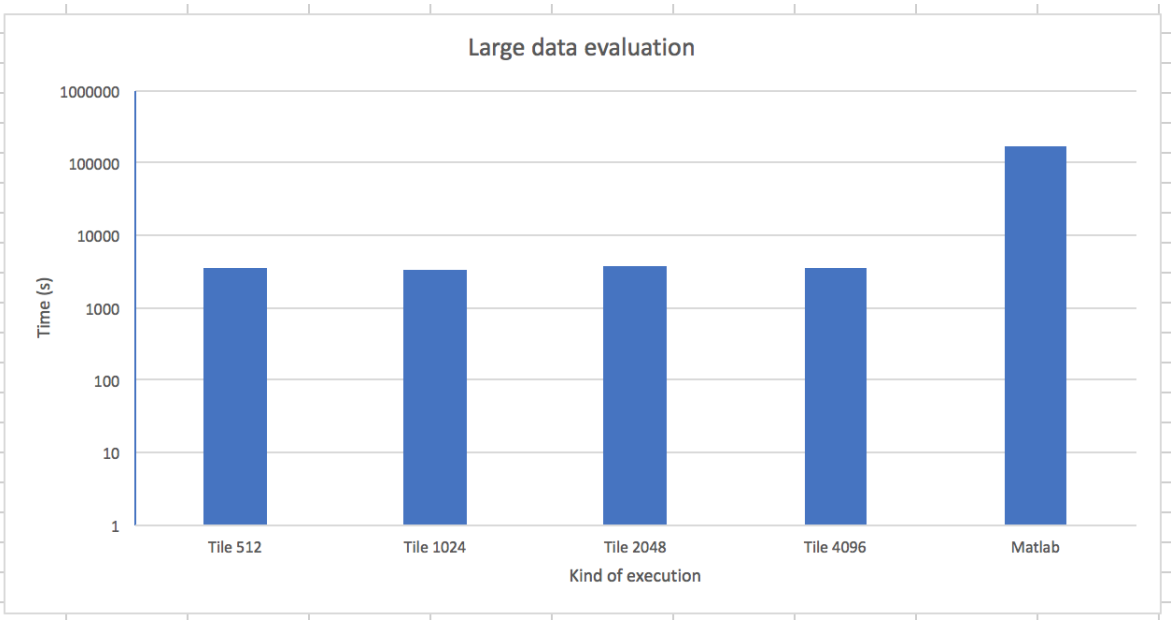

Figura 10-7: Large data Evaluation.

Application performance

The following tables shows the *speedup* obtained as a result of the comparison between the executions of the new application on GPUs and the results obtained from the executions of the application written in *Matlab*.

Small dataset

The best result with the small dataset corresponds 744.99 seconds obtained with a tile size of 1,024. Execution with the previous application had a duration of 3 hours, 10,800 seconds. So that the *speedup* obtained with respect to the small data can be observed in the Equation 10.2.

$$Speedup = \frac{10,800.00\ seconds}{744.99\ seconds} = 14.49 \tag{10.2}$$

Large dataset

The best execution time result with the large dataset corresponds to 3,312.58 seconds obtained with a tile size of 1,024. The execution with the previous application had a duration of 48 hours, 172,800.00 seconds. So the *speedup* obtained with respect to the small data can be observed in the Equation 10.3.

$$Speedup = \frac{172,800.00\ seconds}{3,312.58\ seconds} = 52.16 \tag{10.3}$$

Socio-economic impact

Social impact

The project advisor, author of *pHARDI* and the co-authors maintain the idea that the application is freely distributed, so that any hospital or research group can use this technology and benefit from it.

Performing an MRI implies that the patient should remain immobile in the device for, on average, 30 to 60 minutes. The subsequent reconstruction and processing of the organization of the brain is costly in computation and therefore in time. This can take time, as has already been seen, depending on the size of the data from hours to days.

Thanks to *pHARDI* and the hypothesis of **using GPUs to accelerate the processing of these images**, introduced as the title of this project, we can reduce these times and therefore, be able to offer a diagnosis in a shorter time.

The code can be downloaded from the repository [5] of the ARCOS group of the University Carlos III of Madrid.

Conclusions

The final chapter of the document summarizes the student's conclusions after completing the project and a study of future work that can improve the proposed application. As final conclusions of the proyect, we can state that the hypothesis proposed by the advisor is true, and as shown in Chapter 7, using small dataset, we can obtain a *speedup* of 14.49 and using large dataset result in a *speedup* of 52.16.

As already explained in Chapter 9, the source code is freely distributed for anyone, including any hospital or research center.

Personal conclusions

The main objective of this Bachelor Thesis, as explained in Chapter 1, is to accelerate a magnetic resonance image processor using graphics cards or GPUs, which has been achieved in this project according to the conditions and results described in Chapter 7.

I consider that the most important part of the project is the study and analysis of the state of the art and the acquisition of knowledge of the technologies that were to be used during the project, which is fundamental and basic when carrying out any project. Since it provides A solid base from which the hypotheses of the present Bachelor Thesis start. Another important part of the project has been the analysis of the same, for which the knowledge acquired in the subjects of Software Engineering was used.

The completion of the Bachelor Thesis has enabled the student to consolidate knowledge acquired during the course and assert some knowledge that can only be practiced with the realization of this type of projects. In addition, the student has had the opportunity to use technologies that although explained as theory during the race are not the goal of practices or jobs. In the case of the present project, it can be applied to the programming and use of GPUs.

Other technologies used during the realization were already known by the student. This is the case for example code parallelization techniques by using *OpenMP*, the use of diagram modeling tools or requirements definition.

Future work

This section of the document lists and explains some proposals aimed at improving and complementing the acceleration project of an MRI image processor.

Here are some of the possible future work to be done:

- One of the proposals for future work is to eliminate some dependencies with packages that are used to run the application. This is the case of the *ITK* package which is only used for reading the images in *NIFTI* format. For this, it is desired to develop a library that allows reading this format of images and thus not depend on a package as heavy as *ITK*.
- Use FPGAs for hardware acceleration of executions. FPGAs are a type of hardware device that can be programmed to fulfill a certain function. A future Bachelor Thesis could be based on the FPGA

application to speed up the image processor.

- Graphic interface. It is desired to develop a graphical interface that facilitates the execution of the application and the visualization of the generated images.
- Facilitate installation of the application and the necessary environment. In this way, it would also be possible for people without a high computer knowledge to benefit from the application.

Bibliografía

[1] Arrayfire. http://arrayfire.org/docs/index.htm, (accedido 6 Junio, 2017).

[2] España. Ley orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal, boletín oficial del estado. https://www.boe.es/boe/dias/1999/12/14/pdfs/A43088-43099.pdf, (accedido 6 Junio, 2017).

[3] J. Garcia-Blas, M. F. Dolz, J. D. Garcia, J. Carretero, A. Daducci, Y. Alemán, and E. J. Canales-Rodríguez. Porting Matlab applications to high-performance C++ codes: CPU/GPU-accelerated spherical deconvolution of diffusion MRI data. In *ICA3PP: 16th International Conference on Algorithms and Architectures for Parallel Processing*, pages 630–643, December 2016.

[4] J. Garcia-Blas, J. D. Garcia, M. F. Dolz, Y. Aleman, and E. Canales. https://github.com/arcosuc3m/phardi/blob/master/COPYRIGHT.txt, (accedido 6 Junio, 2017).

[5] J. Garcia-Blas, J. D. Garcia, M. F. Dolz, Y. Aleman, and E. Canales. https://github.com/arcosuc3m/phardi, (accedido 6 Junio, 2017).

[6] HAYS. http://www.hays.es/Guia-Mercado-Laboral-2017/index.htm, (accedido 6 Junio, 2017).

[7] Intel. https://software.intel.com/en-us/mkl, (accedido 6 Junio, 2017).

[8] T. Mathworks. https://es.mathworks.com/products/matlab.html, (accedido 6 Junio, 2017).

[9] NetLib. http://www.netlib.org/blas/, (accedido 6 Junio, 2017).

[10] NetLib. http://www.netlib.org/lapack/, (accedido 6 Junio, 2017).

[11] NVidia. http://www.nvidia.es/object/cuda-parallel-computing-es.html, (accedido 6 Junio, 2017).

[12] NVidia. http://docs.nvidia.com/cuda/nvblas/index.html#axzz4kY9rJ4Be, (accedido 6 Junio, 2017).

[13] REPHRASE. http://rephrase-eu.weebly.com/uploads/3/1/0/9/31098995/d6-3.pdf, (accedido 6 Junio, 2017).

[14] D. C. Sanderson and D. R. Curtin. http://arma.sourceforge.net/, (accedido 6 Junio, 2017).